Aus dem Departement für Informatik
Universität Freiburg (Schweiz)

Verified Network Configuration

Improving Network Reliability

Inaugural–Dissertation

zur Erlangung der Würde eines *Doctor scientiarum informaticarum*
der Mathematisch-Naturwissenschaftlichen Fakultät
der Universität Freiburg in der Schweiz

vorgelegt von

David Buchmann

aus

St.Gallen

Nr. 1613
UniPrint, Freiburg
2008

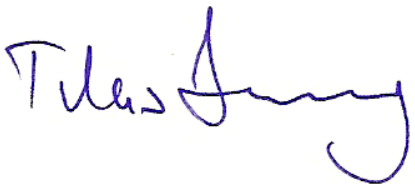
Von der Mathematisch-Naturwissenschaftlichen Fakultät der Universität Freiburg in der Schweiz angenommen, auf Antrag von Prof. Dr. Rolf Ingold (Jurypräsident), Prof. Dr. Peter Kropf (Experte) und Prof. Dr. Ulrich Ultes-Nitsche.

Freiburg, 29. Juli 2008

Der Dissertationsleiter:

Prof. Dr. Ulrich-Ultes Nitsche

Der Dekan:



Prof. Dr. Titus Jenny

Abstract

This thesis describes general concepts of network management and the prototype implementation of a network management system software. The aim of the project Verinec (Verified Network Configuration) is to improve security and reliability in large, heterogeneous networks, while at the same time facilitate their administration. Using a centralised database of the entire network and services configuration, Verinec can validate correctness before configuring the devices. The concept of unit testing used in software development is extended to network configuration. A network simulator is used to run the unit tests against the configuration to ensure operationability of the configuration to deploy.

The configuration is described with the markup language XML. It is abstracted from specific implementations of devices or services. XML technologies are used to translate the abstracted configuration into implementation-specific formats. The translated configuration is automatically distributed using existing remote administration protocols. This avoids introducing errors when applying the tested configuration to the actual network devices.

The aspect of topology discovery and configuration import is also discussed. It allows introducing a network management system in an existing network.

A prototype application has been developed. It is licensed under the GNU Public Licence and all code is available online at `http://verinec.sourceforge.net`.

Keywords: Network Management Systems, Vendor Independant Configuration, Runtime Verification of Configuration, Automated Testing, Reliability and Security

Deutsche Zusammenfassung

Die vorliegende Doktorarbeit beschreibt Konzepte für die Verwaltung und Konfiguration von Netzwerken, sowie die Implementierung eines Computerprogrammes zur Verwaltung und Konfiguration von Netzwerken. Das Ziel des Projektes Verinec (Verified Network Configuration, Verifizierte Netzwerk-Konfiguration) ist es, die Sicherheit und Zuverlässigkeit von grossen, gemischten Netzwerken zu verbessern und dabei gleichzeitig die Verwaltung zu vereinfachen. Dank einer zentralen Datenbank über das Netzwerk und die Konfiguration der Dienste auf den Netzwerkknoten kann Verinec die Korrektheit überprüfen bevor die Konfiguration von Geräten verändert wird. Das Konzept von Unit-Tests aus der Software Entwicklung wird auf Netzwerk-Konfiguration ausgedehnt. Ein Netzwerk-Simulator wird verwendet, um Unit-Tests über die Konfiguration laufen zu lassen. So wird die Korrektheit der Konfiguration für die Geräte sichergestellt.

Die Konfiguration wird in der Beschreibungssprache XML beschrieben. Die Beschreibung abstrahiert von spezifischen Implementierungen von Diensten und Geräten. XML Technologien werden verwendet, um die abstrakte Konfiguration in implementationsspezifische Formate zu übersetzen. Die übersetzte Konfiguration kann automatisiert, mit Hilfe bestehenden Protokollen zur Fernwartung, verteilt werden. Damit wird verhindert, dass beim Übertragen der getesteten Konfiguration auf die Netzwerkgeräte neue Fehler entstehen.

Weiter werden die Themen Netzwerkanalyse und Konfigurationsimport angesprochen. Dies erlaubt es, ein System zur Verwaltung des Netzwerkes in einem bestehenden Netzwerk einzuführen.

Acknowledgements

This PhD thesis has been written in the context of the Verinec project at the University of Fribourg, Switzerland. The Verinec project was supported by the *Swiss National Science Foundation* under the grants number 200021-100645/1 and 200020-108056/1. Preliminary research on the concept has been done by Simon Chudley in the project NetSim [Chudley and Ultes-Nitsche, 2002].

I want to thank Professor Ulrich Ultes-Nitsche for his inspiring support and help with the Verinec project and this thesis. Interesting suggestions for the direction of research came from him. He helped me going on and encouraged me to submit papers to various conferences, and corrected my spelling errors in them.

My sincere thanks go to Dominik Jungo. We worked together on this project and pushed each other to finish tasks to continue our work. He primarily worked on the verification of configuration, while I focussed on configuration definition and adaptation to actual implementations. But there were many design decisions we had to take together. We could take them on a technical basis, with neighter of us having to tough it out, for which I am very grateful.

I also want to thank the Bachelor and Master students who wrote their theses in the Verinec project. Their various implementation fragments contributed to the Verinec application are credited in the according sections of this thesis. They are, in alphabethical order, Patrick Aebischer, Geraldine Antener, Robert Awesso, Christoph Ehret, Jason Hug, Renato Loeffel, Shewangizaw Mengistuabebe, Martial Seifriz, Damian Vogel and Nadine Zurkinden.

My thanks also go to Karin Baasch for proofreading this dissertation, detecting an awful lot of grammar and spelling errors. Last but not least, I want to thank my friend Christine Scheidegger for her support and encouragement during all that time. And for proofreading the complete thesis and pointing out where it was not readable enough.

Contents

Chapter 1

Introduction

Seasoned Cisco administrators can give you a hot tip on how to reduce outages in the network after a configuration update on a router. You give the new filter list a different name than the name of the list currently used. While having a ping running in one window to see whether connectivity is still up, you tell the interface to switch to the new filter list. If the ping stops getting responses, switch immediately back to the old list and try to figure out what the mistake was.[1]

Does this sound complicated and a bit awkward? Would it not be much better if your configuration tool could just tell you whether your configuration is correct before the configuration of the router is modified at all? This tip was distributed on an Internet mailing list for Cisco administrators. And as long as network management tools do not evolve, it is actually quite a good idea.

Networks play a crucial role in modern enterprises. Even short periods of failure can result in loss of revenue. In the area of dependable systems not just money, but human lives are at stake. More and more enterprises need networks to be available around the clock without interruptions.

Building a reliable computer network is a difficult task, the larger the network the worse it gets. Planning and configuring productive networks without specialised tools is clearly not a good solution. Beyond a certain size, it becomes practically impossible to keep track of the network manually. The risk of human error increases rapidly, calling for intelligent solutions to support administrators. Research in network management has been examining the tasks, challenges, solutions and tools for networks. A wealth of tools has been developped to avoid unnecessary manual chores and to increase security.

Various papers report that configuration errors are one of the most important sources of service interruptions, e.g. [Mahajan et al., 2002]. [Oppenheimer et al., 2003] analysed large Internet sites and found that errors made by the human operators are the most important cause of failure. While the Internet sites they examined were quite big, their network structures are not nearly as complicated as a typical network of a large company. In company networks, the variety of protocols as well as users at different levels of trust are bound to make operator errors even more likely. Configuration errors are a class of error that could be avoided if the Network Management System (NMS) could understand the intentions of the administrator and assess the requirements of the network. To put it in other words, an NMS should be able to verify the semantic correctness of the configuration prior to applying it to the devices.

As will be discussed in Section 2.3, one source of network service interruptions is invalid configuration sent to nodes. Typical NMS ensure the correct syntax of configuration files. Advanced NMS support policy frameworks to further control how a device may be configured. However, the question whether services in the network will really be able to operate cannot be answered in advance by using only policies and syntax checks.

[1] The author found this tip while surfing the web for Cisco documentation. The original source is unfortunately no longer to be found.

1.1 Goal of this Thesis: Improving Network Configuration

In the Verified Network Configuration (Verinec) project – supported by the swiss national science foundation – new concepts for network configuration management are explored. This is a joint work of Dominik Jungo and David Buchmann, under the supervision of Prof. Ulrich Ultes-Nitsche. We tried to combine several technologies into a prototype application that understands configuration and can test the correctness of configuration. All code of the prototype application is under an Open Source license and available over the Internet.[2]

Misconfiguration - or how to avoid it - is the main topic of the Verinec project and hence of this thesis. Dominik Jungo focussed on the verification of configuration, while David Buchmann put most work into configuration definition in XML and adaptation of the configuration to actual implementations. But by nature of the collaboration, both contributed to all aspects of Verinec.

One key technology for Verinec is **network simulation** of each service on a detailed scale to automate the testing of requirements. With the automated testing, an administrator can simply specify web access as a requirement and the simulation will find out if any of the involved services are not reachable. Using simulation test cases similar to unit tests in programming, it is possible to check the configuration for problems after every configuration edit. Moreover, mistakes can be found without interrupting services in the production network. If the whole configuration is verified before being applied to configuration, misconfiguration will be reduced.

Many NMS have been developed and are commercially available, as the overview of existing concepts in Chapter 2.4 shows. The lack of an omnipresent interoperability standard has resulted in companies using a - more or less integrated - combination of several systems that support only a subset of the available devices and the tasks required to configure the network. The Verinec project is designed to be easily extensible to handle every kind of network and services configuration. The dependence on a specific remote management protocol is avoided by using an extensible framework for configuration distribution. Current NMS research (see Chapter 2) often focuses on distributed concepts, because they are more scalable and flexible. For the simulation however, it is necessary to know about all of the configuration. In the Verinec project, the network and system configuration is stored in one central place. The internal representation of the network is independent of specific products. It is focused on the common functionality of each service. The internal, **abstract configuration** is expressed in Extensible Markup Language (XML). Special care is taken to avoid redundancy in the XML configuration documents.

Avoiding manual operations saves work and increases security at the same time, as repetitive tasks are particularly error-prone. What's more, this connects strongly to the concept of only letting verified configuration onto a device. The verified configuration must not be modified accidentally - or on purpose - in the process of applying it to the devices. For this reasons, there is a need to **automate the distribution of configuration**. Abstract configuration must be translated for the concrete devices and the devices must be automatically updated to the new configuration. To **import existing networks**, traffic analysis and configuration file parsers are considered.

Verinec is currently limited to *configuration* management. It does not monitor resources, create alarms or provide usage statistics. There are sophisticated tools for the task of monitoring a network, for example OpenNMS or OCS Inventory.[3] The motivation behind the Verinec project was to **explore a new approach in order to improve the quality of network configuration**. Verinec verifies the configuration in its abstract XML representation and automatically applies it to the network devices.

[2] Download the Verinec prototype at `http://verinec.sourceforge.net`

[3] See `www.openms.org` and `www.ocsinventory-ng.org/`

1.2 Pre-publications

While working on the Verinec project, we were able to publish several papers. They are listed here in chronological order. The main author of each paper is listed first, the other contributors are mentioned too.

The Role of Simulation in a Network Configuration Engineering Approach
Dominik Jungo, David Buchmann, Ulrich Ultes-Nitsche: ICICT 2004, Cairo, Egypt.
[Jungo et al., 2004] presents the approach of simulation to see whether network configuration will work properly. The idea of automated translation is mentioned already.

A Unit Testing Framework for Network Configurations
Dominik Jungo, David Buchmann, Ulrich Ultes-Nitsche: MSVVEIS 2005, Miami, USA.
[Jungo et al., 2005] develops the idea of simulation for verification to the notion of network test cases. Input events and expectations on the output can be combined to automate tests on network configuration.

Automated Configuration Distribution in Verinec
David Buchmann, Dominik Jungo, Ulrich Ultes-Nitsche: ICETE 2005, Reading, UK.
[Buchmann et al., 2005] describes the translation process. Restrictors are presented to handle incomplete translation due to limited capabilities of the target implementation. The distribution process as also discussed.

CLiXML - Testing of semantic properties in XML documents
Dominik Jungo, David Buchmann, Ulrich Ultes-Nitsche: MSVVEIS 2006, Cyprus.
[Jungo et al., 2006] introduces the Constraint Language in XML and presents the implementation openclixml. This rule based constraint language is used in Verinec for the network test cases.

Environmental Acquisition in Mobile Network Simulation
David Buchmann, Dominik Jungo, Ulrich Ultes-Nitsche: WINSYS 2006, Setubal, Portugal.
[Buchmann et al., 2006] discusses the application of environmental acquisition to network configuration. Configuration fragments can get values based on their context and not only on their lexical structure. This is used to avoid redundancy in configuration.

A role model to cope with the complexity of network configuration
David Buchmann, Dominik Jungo, Ulrich Ultes-Nitsche: INOC 2007, Spa, Belgium.
[Buchmann et al., 2007b] examines possibilities to further reduce redundancy in configuration. Role models can be used to encapsulate configuration templates. Environmental Acquisition can be used to parameterise the templates for the place they are used in.

Assessment of code quality through classification of unit tests in VeriNeC
Dominik Jungo, David Buchmann, Ulrich Ultes-Nitsche: AINA 2007, Niagara, Canada.
[Jungo et al., 2007] further examines unit tests and proposes to distinguish different classes of tests. These help to qualify tested code / configuration segments as being to restrictive, to permitting or violating some other class of tests.

Improving Network Reliability by avoiding Misconfiguration
David Buchmann, Dominik Jungo, Ulrich Ultes-Nitsche: DRCN 2007, La Rochelle, France.
[Buchmann et al., 2007a] explores the possibilities offered by network simulation. Simulation cannot only be used to test one configuration, but also to assess failure reliability of networks. Failure of any combination of nodes can be simulated to see whether the network would still operate when some nodes fail.

1.3 Terminology

The Management of computer networks has been researched for some time. In order to avoid misunderstandings, this section clarifies the usage of some key terms within this paper.

A first term is *network*. In computer science, a network refers to a group of computing devices that are enabled to exchange data over some medium. Every node of the network is a *network element*.

The term *network management* has different meanings. It does not necessarily mean active operations that configure a network, but can also refer to monitoring network usage, gathering statistics and provide error notifications. In the context of Simple Network Management Protocol (SNMP) for example, the term management is used in a broad sense. SNMP specifies how to gather information about the configuration and operational statistics as well as active changes to the configuration. While configuring network elements and building statistics about the network are related, they are separate tasks. Unfortunately, SNMP implementations often only implement the status indication part and lack support for active configuration. The term network management within this paper will refer to the complete set of tasks that keep a network under control, including modifying configuration. The Verinec project is focused on the *configuration* part of network management.

What kind of networks do we manage? The term *Network Management System (NMS)* is sometimes used for monitoring and restricted to layers 2 and 3, usually for router management. This notion is typically found in documentation coming from Cisco. Sometimes the term *Network Configuration Management* is used to explicitly declare that a system is not just designed for monitoring but also for actively manipulating configuration. To distinguish from router management, some authors use the term *Network and Systems Management (N+SM)* [Anderson and Burgess, 2001]. This term indicates that not only network devices but also server machines are managed. In the Verinec project, all layers of the network are taken into account and both networking hardware and upper layer services on server computers can be configured. In this thesis the term *network management* means the administration of all parts and services of the network.

An area close to configuration management is *software management*. From an administrator's view, the information what software packages are installed on a system can also be considered a configuration issue. Additionally, updates for critical security fixes are an important aspect of security management. Normal software updates also have to be automated in larger networks. Software management is not taken into consideration in the Verinec project.

To avoid confusion between the human operators of a network and SNMP terminology, this thesis will follow the suggestions of [Martin-Flatin et al., 1999]. A *manager* is the station used to send commands to *agents*, which are dumb applications running on devices that have to be configured. The human responsible for the network on the other side is called *administrator*. The word *agent* is used differently in software engineering. For this thesis, the term agent is to be understood in the sense of the manager-agent paradigm, it is thus simply carrying out what it is told by the manager. Agents in the sense of co-operating agents are always named *intelligent agents*.

1.4 Related Work

Network configuration management has evolved with the growth of networks. There is a wealth of commercial NMS available, but unfortunately often targeted at a subset of network management or only devices from certain manufacturers. Hardware manufacturers like Cisco [Cisco Systems, 2007b] or Check Point [Check Point Inc., 2007] focus on software to manage their own hardware. These applications, as well as the vendor-independent SPLAT [Abrahamson et al., 2003], are suited to administrating large numbers of routers and are mostly used by service providers. Several specialised companies concentrate on network management applications, for example [Aprisma, 2006, Voyence, 2006, Intelliden, 2006]. Tools for specific tasks are available as open source projects, for example Splat [Abrahamson et al., 2003], Edge [Caldwell et al., 2003] or ONA [Campbell, 2005]. Some of the big players in the computer industry also offer their own management systems, amongst them IBM Tivoli NetView [IBM, 2007] which builds on the CORBA framework and HP OpenView [Hewlett-Packard, 2007] based on SNMP. The later systems focus on error reporting and building

statistics. They have been criticised for their lacking flexibility. [Garschhammer and Schiffers, 2005] for example states that NetView and OpenView lack support for inter-organisational co-ordination. Organisation and enterprise networks are a lot more heterogeneous than service providers' infrastructures.

Concepts and theoretical foundations for network management came later. Paul Anderson from the University of Edinburgh lead the development of LCFG [Anderson and Scobie, 2002] and published numerous papers on network management. Marc Burgess from Oslo University developed cfengine and formulated the vision of "computer immunology" [Burgess, 1998]. Sanjai Narain proposes to use formal verification for network configuration [Narain et al., 2003]. The Netopeer [Lhotka and Novotny, 2007] project around the Czech researchers Ladislav Lhotka and Jiri Novotny uses an approach similar to that of Verinec with abstract configuration and translation, but focuses on routing and firewall configuration. There is also a research program by the European Union called DESEREC (DEpendability and Security by Enhanced REConfigurability). The aim is to improve resilience of critical Information Systems. One part is a system similar to Verinec that shall be able to model and simulate networks. DESEREC also wants to integrate detection of unwanted behaviour of the network and combine it with computer aided response measures [Pérez and Bruyère, 2007].

In the context of growing complexity due to integration and the concept of "Grid Computing", new management concepts are needed. [Garschhammer and Schiffers, 2005] discusses to what extent existing management solutions can be re-used in dynamic IT systems and what has to be rebuilt from scratch. Some authors hope to automate network management completely, for example [Kephart and Chess, 2003]. A discussion of the challenges in building such a solution is given in [Herrmann et al., 2005]. However, their conclusions confirm that there is still room for improvements in the more classical network management before self-management can become a reality.

1.5 How this Thesis is Organised

The basic theory of network and service management is discussed in Chapter 2. Concerning the Verinec project, the authors have only implemented support for some of the more important network services. A general introduction to the operation of a network focussing on those services is given in Chapter 3. This foundation serves as an introduction to Part II on different aspects of network management. Part II considers more aspects than have been implemented in Verinec. The actual implementation is discussed only later in Part III. As a motivation for the Verinec project, Chapter 4 presents a series of use cases typically encountered in network management. Verification of the network configuration is discussed in Chapter 5. Chapter 6 presents the automated distribution of configuration to the end devices. For practical reasons, the Verinec project also considered possibilities of importing existing configuration. Chapter 7 details the research on this aspect.

Based on the previous theoretical chapters, Part III presents the Verinec application in Chapter 8. The architecture of the application is explained and how configuration data is expressed and presented in the graphical network management application. Chapter 9 illustrates the application of Verinec in a case study. The last chapter contains conclusions and an outlook on further possibilities. The appendix contains a list of acronyms and the bibliography.

Because this thesis covers different aspects of network management, the discussion of related work is not concentrated in one chapter. It is included in the chapters on network management, verification and the adaptation of configuration.

Part I

Network Theory

Chapter 2

Network Management

The aim of network management is to provide a reliable and secure network. To achieve this, the network needs to be planned soundly, set up correctly and then constantly monitored and maintained. Keeping the network documentation accurate and up to date after modifications is crucial to keep an overview of the network. Automated support is required to track the configuration of the enormous number of devices present in a large network.

Administrators of large networks face a number of tough challenges. Requirements and expectations constantly change. Technology evolves and each new generation of devices is loaded with more features. As soon as vulnerabilities in operating systems and applications are discovered, fixes need to be distributed. What's more, every device comes with its own administration tool or web interface. In heterogeneous environments, the administrator has to learn various different concepts and user interfaces to configure similar devices. When company structures change, new network parts have to be integrated, sometimes leading to major changes in the network structure and resulting in further heterogeneity. Even when no urgent problem becomes manifest in the network, it has to be constantly monitored in order to detect any unexpected behaviour and fix issues before they affect the users. The amount of information collected and the number of possibly false or irrelevant alerts requires automatic filtering.

And while one attempts to fulfil all these requirements, costs have to be kept in check. Goals have to be reached with a limited budget and finite human resources. While the infrastructure is crucial for many companies to operate, it usually does not generate revenue by itself. Expenses for additional hardware must be justified and unnecessary investments should be avoided. A method for calculating the costs of network problems is demonstrated in Section 2.1, making it easier to justify spending money on network security. Besides, a complicated infrastructure also requires more administrators, leading to higher labour costs as well. On the other hand, a well-structured network not only improves reliability, it also frees resources in the IT department for tasks other than trying to keep up with incoming problems.

Various standards have been developed to help companies set up a responsible network management. In economic sectors dealing with particularly valuable customer data, network security is not only important to the company itself, but also for the customers. Some countries also compel companies to establish a security concept by law, for example through an obligation to protect the privacy of customers.

The main focus of this thesis lies not on the organisational processes but on the handling of network configuration. In order to describe the context surrounding network security, this chapter nonetheless discusses some aspects of organisational interest. The following section discusses how to calculate the costs of network outages. Hopefully this will show why a reliable network is also important from an economic point of view. The next section gives a summary of common management standards. After this, the key aspects of network management are examined on a more technical level. The chapter ends with an overview of current research on network configuration and management. Network management principles are classified and different user groups are defined.

2.1 The Cost of Network Downtimes

As mentioned in the introduction, expenses to improve the network infrastructure need to be justified, as the network does not directly generate revenue. Some research has been done to quantify the losses resulting from network problems, for example [Patterson, 2002]. Efforts to improve the network and to make it more fail-safe have to be weighed against the estimated costs resulting from downtimes. Depending on the line of business a company is in, the term "downtime" can relate to different problems. If the business sells goods online, the most important network part is the web server infrastructure. For some companies, employees' Internet access is the most important thing. Others are more concerned about communication in their intranet, or employees might not be allowed to spend time on the Internet at all. [Patterson, 2002] introduces a concept that covers all of these cases. The costs for one hour of downtime is estimated as the costs for employees per hour plus the average revenue per hour. Both costs are reduced to the percentages estimated to be affected by the outage. The formula can be noted as:

$$\text{Costs 1 hour downtime} = \text{Empl. costs/hour} * \% \text{ affected} + \text{Avg. Rev./hour} * \% \text{ affected}$$

This formula takes into account both the lost work of employees and the revenue lost during that time because customers could not place orders. Employee costs are the total costs for salaries and benefits averaged to one hour. Average revenue is the total revenue generated by the company, also averaged to express the revenue generated in one hour. The fraction made up of employees and revenue affected by an outage is an estimate based on the actual company structure and the network layout. Network simulation might help to improve these estimations by assessing the impact of various possible failures.

The estimate is open to discussion. On the one hand, employees could do something else during the downtime and customers could try to order again later. On the other hand, downtimes can even result in employees having to work overtime, raising the costs even more. Too many downtimes also cause image problems, customers may choose to do business with a competitor instead. Often the time of day during which an outage occurs is also important, but some problems specifically occur under high load. Applying worst and best case scenarios might be useful.

[Patterson, 2002] provides three sample calculations. First he examines the Electrical Engineering and Computer Science department at U. C. Berkeley, where he is employed. In spite of the fact that a university does not really generate revenue, one can still calculate the costs of lost work. On the assumption of a 10 hour working day, employee costs are $14,780 per hour during term and $19,170 during the summer, since many paid projects are worked on during this time. If file and mail servers have a reliability of 99%, one can expect 87 hours of downtime per year. If half of them affects 50% of the employees, the annual costs in lost productivity are between $250,000 and $300,000.

Amazon, as a big player in Internet sales, generates most of its revenue online. An outage is likely to affect a lot of their revenue, but they also have a considerable number of employees. The calculation is based on business information detailed for 2001 by Amazon. The revenue was $3.1 billions, distributed evenly over the year, this is $353,900 per hour. Assuming an average salary and benefits of $85,000 per year and 5 days of 10 hours of work each week, their 7744 employees cause hourly working costs of $258,100. An outage during working hours, blocking 90% of the revenue and 90% of the employees would cost $550,000 per hour. Obviously, Amazon has to prevent even short downtimes and can justify spending large sums of money on the security of their systems.

The third example is Sun Microsystems. Patterson assumes that most of their revenue is generated by sales persons rather than directly in online shops. The sales force, however, needs email and other Internet technologies to contact its clients. Again with the business data of 2001, he estimates that an outage affecting 10% of the revenue and 90% of the employees would cost about $825,000 per hour.

Finally, Patterson mentions that frequent problems with the network can have an impact on the company culture. When morale suffers and IT is blamed for all kind of problems, productivity can decrease for longer periods than the actual outage. If departments start to engage their own IT staff because they do no longer trust the IT team, this results in direct costs. The concept presented by Patterson is used by companies to convince customers to invest in network reliability

improvements.[1] There is even an online calculator available.[2]

A study by [Systimax, 2007] confirms the financial significance of network downtime. Almost 1500 information technology professionals answered their survey on network usage and importance. The study indicates that various economic sectors differ in their network requirements. In financial companies, for example, almost all of the employees depend on the network, while in the real estate or construction sectors, usually only 50% need the network. Accordingly, downtime is considered to have a major cost impact on the financial sector, while real estate and construction are able to deal with some downtime. A white paper by [Performance Technologies, 2001] cites other studies that confirm the loss of millions of dollars due to network outages. They noted a concentration of work group servers that stood near their users on centralised server farms. The reason for this is that a server farm is easier to manage both for physical access and for backup systems. On the technical side, the performance of networks has increased and powerful servers are available that can handle large loads. The downside is a higher dependency on the network. Problems with the core network tend to affect a higher proportion of users.

2.2 Network Management Standards

This section discusses various standards for secure network management, coming from different contexts. As network management is a complex area, there is no single obvious solution. Some standards emphasise the management perspective of a company while others focus on the tasks of the IT department. Figure 2.1 depicts the landscape of management standards and concepts discussed in this chapter. The Sarbanes-Oxley Act (SOX) is a actual law in the United States concerning (amongst other matters) network security. Standards defined by the ISO are used as reference points throughout the industry. FCAPS is based on a technology-centred view, which makes it easy to adopt but misses the more complex levels of management. International Organization for Standardization (ISO) applied FCAPS in their famous Open Systems Interconnection Reference Model (OSI) for networks [Parker, 2005]. The ISO 2700x defines a set of general standards for network management. The Payment Card Industry Data Security Standard (PCI) standard is included as an example of a domain specific agreement. ITIL and CobIT are best practises references in the form of a series of books.

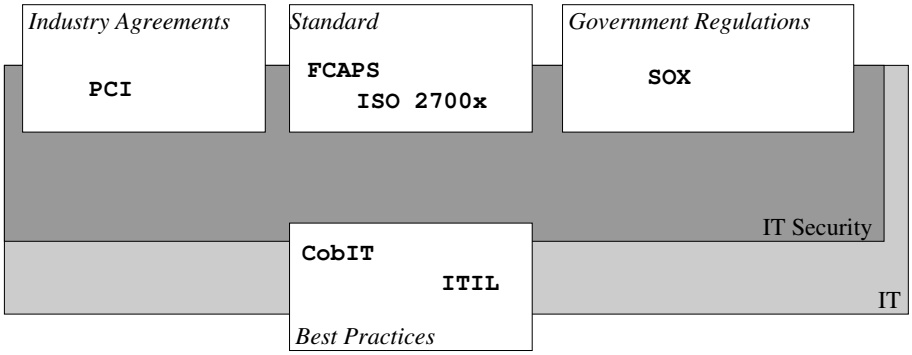


Figure 2.1: Management Standards.

Each standard divides network management into a group of different tasks to keep in mind in order to maintain a reliable network. The different standards are not contradictory, but focus on different aspects and choose different re-partitions of the tasks. The focus of this section lies on how the tasks are partitioned.

2.2.1 FCAPS

The acronym FCAPS stands for Faults, Configuration, Accounting/Administration, Performance and Security. All of the five aspects of network management have a kind of life cycle from anal-

[1] Seewww.netforcement.com/network_monitoring/network_downtime.asp.

[2] www.baker.ie/bakerwatch/calculator.html.

ysis to planning, implementation, monitoring and back to analysis. An introduction to network management using FCAPS is given in [Castelli, 2001].

Fault management has to recognise, isolate and correct faults. It has to monitor resources and trigger alarms when there are failures. After failures, the network functions have to be restored as soon as possible. This might be done using temporary solutions, until the root cause for the fault can be fixed.
Alarms and other events have to be handled in such a way that administrators are not disturbed by false alarms and yet are provided with a good interface view of all important details.

Configuration management deals with the installation and configuration of the network components. It has to keep track of configuration versions and provide backups. The inventory describes what hardware is used and where it is located. It should simplify configuration and automate as much as possible. Autodiscovery of network elements and configuration can help to achieve this goal.

Accounting/Administration tracks the usage of services. FCAPS comes from the telecommunications world, where billing customers for the usage of services is an important factor. But the idea can be used for packet-based networks too. For networks that are not billed based on detailed usage, the A is seen as administration instead of accounting. Administration focuses on controlling quotas and handling user accounts, passwords and access permissions.

Performance management collects data about network parameters, generates reports and analyses them. For resolving bottlenecks, the administrator needs to know which network elements are the limiting factor, otherwise he might spend a lot of money on unnecessary improvements. Trends in network usage are assessed in order to identify upcoming bottlenecks and prevent other problems. Alarms can be triggered for fault management if the usage of a monitored resource increases dramatically.

Security management has to identify risks and solve them. It has to ensure legitimate use, maintain data integrity and confidentiality. To allow for auditability, security relevant operations need to be logged in a reliable fashion. Security involves extra management tasks, extra processing, special hardware and can be inconvenient for users at times. The benefits are not directly visible, but deficient security management can manifest in severe problems. Examples are service outages, theft of intellectual property, confidential data or computing and traffic resources, and sometimes legal issues.

2.2.2 ISO 27001

The full title of the standard is "Information Security Management - Specification With Guidance for Use". The aim is to describe an Information Security Management System (ISMS) and allow for audit by third parties. The ISO wrote 27001 based on standards defined by the British Standard Organisation (BS). The next section mentions the old BS standards as well to clarify the transition.

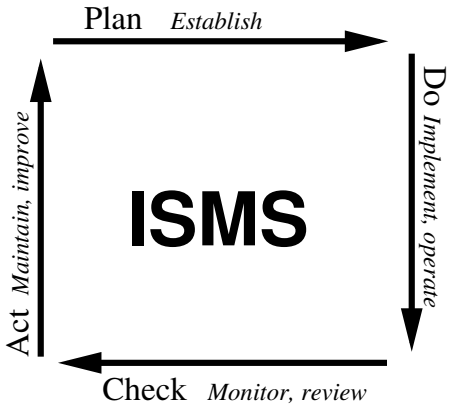


Figure 2.2: The Plan-do-check-act (PDCA) model. Figure inspired by the 17799 wiki [Sweeney et al., 2007].

[Sweeney et al., 2007] manage a wiki to explain the family of BS and ISO security standards. ISO 17799 describes a "code of practice" for network management (adopted from BS7799-1). The standard refers to the risk assessment provided by an ISMS, as is described in ISO 27001 (replaces BS7799-2 and is harmonised with the important ISO management standards 9001 and 14001). Certification of enterprises is done according to the ISO 27001 – the ISMS – and not according to the code of practice. There are plans to rewrite ISO 17799 as 27002 and build a suite of 2700x norms for security management.

ISO 27001 describes an ISMS to support the continual improvement of network security. The ISMS is built using the PDCA model: Plan-do-check-act, depicted in Figure 2.2. The process of securing a network from a management point of view is divided into 6 steps.

- Define the scope of the ISMS.
- Define a security policy.
- Undertake a risk assessment/analysis.
- Manage the risk.
- Select control objectives and the actual controls to be implemented/applied.
- Write a Statement of Applicability. It has to specify which controls – usually taken from ISO 17799 – are linked to which risks.

The ISO standard describes network security on a management level, leaving implementation to other standards and best practises agreements.

2.2.3 ITIL Configuration Management

Another approach is the IT Infrastructure Library (ITIL). ITIL is defined in a series of books to aid the implementation of IT service management. The standard was defined by the UK governments Central Computer and Telecommunications Agency (CCTA) in the 1980's. The goal was to create standard processes according to best practices for the recurring problems of IT service management organisation. Meanwhile, CCTA has been renamed to Office of Government Commerce (OGC) and has published a second version of ITIL. The standards proposed in ITIL are used throughout the world now.

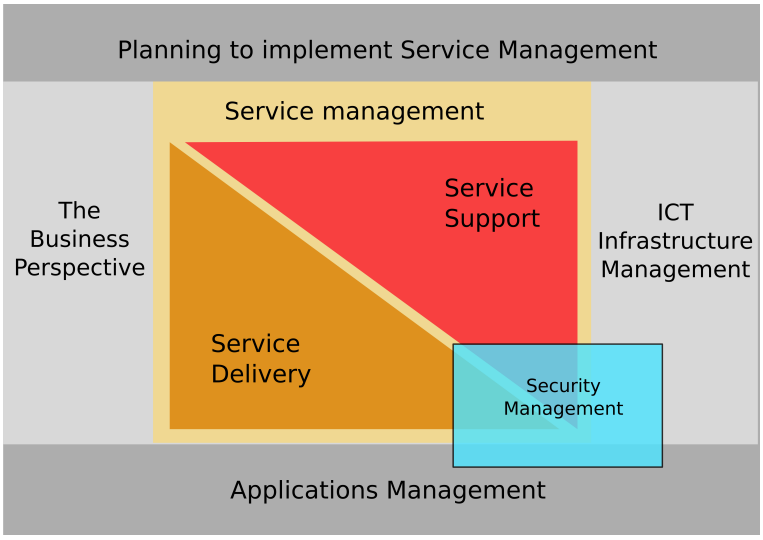


Figure 2.3: Information Technology Infrastructure Library ITIL.

The library is divided into seven 'sets': Service Support, Service Delivery, Planning to Implement Service Management, Applications Management, ICT Infrastructure Management, The Business Perspective and Security Management. The sets complement each other, but a company can use ITIL without implementing all seven sets. Figure 2.3 shows the sets and relations between them. Notably Security Management is considered a separate set which affects and controls all other sets.

Within the context of this thesis, Service Support is the most important set. It contains the process of Configuration Management, amongst other things. This process keeps track of the individual Configuration Items (CI) in a system. Basic activities of Configuration Management are:

Identification: Collecting all information about CI's, their ownership, relations to other CI's and different versions.

Control: Making sure that only authorised staff can change a CI and that each change is documented and justified.

Status Accounting: Keeping track of all versions of CIs. This also informs on whether configurations are, for example, 'live', 'under development', 'testing' or 'outdated'.

Verification and Audit: Periodically ensuring that the recorded information about CI's is correct and up to date.

ITIL is a broader standard than FCAPS, directly taking the business side of the network management and the planning process into account. For further reading on ITIL, please refer to the ITIL web site at `www.itil.co.uk`.

2.2.4 Sarbanes-Oxley Act and CobIT framework

While PCI compliance (discussed in the next section) is based on the initiative of a private business organisation, Sarbanes-Oxley (Sox) is a federal law in the United States. [Sarbanes and Oxley, 2002] The act was passed in response to incidents of major accounting scandals in large American companies, with the aim of protecting investors by improving the reliability of company statements and by impeding insider deals. It is applied to all companies listed in American stock exchanges. Its name derives from its main sponsors, Senator Paul Sarbanes and Representative Michael G. Oxley.

As the important business data is almost always stored in computer networks, it contains quite detailed rules for information security - which is why this act is worth mentioning here. Contrary to FCAPS, which is no more than a suggestion aimed at helping companies with security, and PCI compliance which is enforced by contracts between credit card companies and merchants, Sox is enforced by law in the USA. Non-compliance can result in legal consequences.

Describing the entire Sox act here would be beyond the scope of this thesis. This description is limited to the aspects having an impact on computer networks. The overseeing board founded to control the implementation of Sox recommends using the Committee of Sponsoring Organizations of the Treadway Commission (COSO) framework to do audits ensuring proper internal handling. COSO is a private initiative with the goal of reducing the number of forged financial reports and other frauds. The COSO ideas have become part of the CobIT management model. The framework identifies five components:

Risk Assessment: Prior to implementing controls, IT staff has to determine and understand the areas of risk which could affect financial reports. These areas of risk are the basis for the other four components of the framework.

Control Environment: Company morale is important enough to be granted a component of its own. If employees understand the importance of security and feel encouraged to take responsibility, security policies will exist not only on paper but people will also adhere to them.

Control Activities: Detailed policies have to specify security protocols and other technical requirements for each project. Projects must be approved and authorised and management has to ensure that policies are followed on all organisational levels.

Monitoring: Regular auditing processes should be undertaken, both by internal staff and by experts independent of the company. Regular activities as well as separate evaluations have to be planned. Of course, problems detected by the monitoring process should be reported in a clearly defined way, leading to actions undertaken to solve them to ensure continuous amelioration.

Information and Communication: Important information concerning security must be distributed effectively within the company. Management is also responsible to see that every employee is aware of his or her role in the security system. The communication must also work with customers, suppliers and shareholders.

The emphasis is clearly on the management side and less on the technical measures in the IT department. It is interesting that the focus lies on the employees' understanding of the purpose and relevance of security measures. Technical measures applied without the understanding of the staff often do not improve security. People start to find ways around the impediment of security procedures, sometimes resulting in even bigger security holes. The lesson of CobIT/COSO is that security has to work together with the staff of a company, not against them.

2.2.5 Payment Card Industry Data Security Standard

MasterCard and VISA designed the Payment Card Industry Data Security Standard (PCI), which is now enforced by them as well as other credit card companies for all members, merchants and service providers. Non-compliance with the PCI standard can result in high fines should incidents occur. According to [Gertner, 2005], this standard was developed in reaction to repeated security breaches in banking networks, leading to customer data being compromised or stolen.

The standard defines six areas that a company wanting to do business with credit cards has to comply with:

Build and Maintain a Secure Network requests that firewalls are installed and maintained. All traffic from untrusted networks has to be blocked to avoid any unnecessary connectivity. Additionally, default or vendor-supplied passwords must be changed before a system is put into use.

Protect Cardholder Data says that any stored data has to be protected, also from company employees with the exception of those who really need access to it. If cardholder data or other sensitive information has to be transmitted over public networks, it must be encrypted.

Maintain a Vulnerability Management Program means that anti-virus software has to be used and updated regularly. The organisation must have a concept for updating applications bought from third parties when security relevant patches become available and need a similar concept for their home grown applications as well.

Implement Strong Access Control Measures requests that only users really needing the information should be allowed access to relevant data. Users having access to the data must be unambiguously identified, both for restricting their access and for monitoring their actions. Physical access to the servers storing cardholder data has to be restricted to avoid physical intrusion and theft of data.

Regularly Monitor and Test Networks requires tracking and monitoring all access to network resources and cardholder data. Security systems and processes have to be tested regularly. A secure logging of all actions, preventing everybody from deleting entries for some period of time allows one to trace who accessed which data.

Maintain an Information Security Policy means that the IT department and management have to define and enforce a policy throughout the company. The policy should address all areas of the PCI standard and set clear rules and regulations for the users.

The standard does not define compliance for products, but for the members, merchants and service providers. It is not enough to buy a specific firewall; the IT department has to fulfil the specified requirements and is reviewed by external compliance testers.

While the standard was developed specifically for the credit card industry, it seems to be a good security standard for any kind of organisation dealing with confidential data. Unlike FCAPS and ITIL, PCI does not define the complete network management, but only the security aspects of it; these are defined in a very precise and detailed manner. More information on the standard is available at the official web site `www.pcicomplianceguide.org`.

2.3 Key Aspects of Network Management

In the previous section, a number of security management standards were introduced. The standards describe the whole management process and thus operate on a high level of abstraction.

An NMS aims at keeping control of a large network setup. This section looks at the more technical aspects of ensuring correct and secure configuration in a network. Several tasks have to be accomplished:

Inventory: *Automated network discovery and import of configuration.* So far, it remains difficult to keep an accurate map of a network. Changes often occur in emergency situations, when something has stopped working. It is quite useless to work with anything but an accurate map. Automated discovery can help a great deal, comparing the existing network with possibly outdated records.

Verification: *Test correctness of the network configuration to detect mistakes prior to changing the productive network.* Configuration mistakes on productive devices either lead to immediate service interruption or are like ticking time bombs. The defence against attacks can be compromised or service interruptions can occur later, when the use of the system changes. Configuration should never be applied to a productive device untested.

History: *Keep track of configuration changes and archive old configurations.* Tracking changes has two goals. One is to revert to old versions known to work when an update breaks something or to simply have a backup for when a configuration is inadvertently deleted. The other goal is to conduct audits and render the person who introduced errors accountable.

Automation: *Devices should be configured directly by the NMS. Repetitive tasks for administrators should be avoided.* If administrators have direct access to the devices to be configured, maintaining history and verification records depends on their willingness and discipline. Auditing configuration management is not possible without exact records. Automating repetitive tasks means reducing the redundancies of the configuration an administrator has to manage, avoiding mistakes while cutting down on unnecessary work.

These tasks will be revisited in later chapters. Verification is discussed in Chapter 5, automation in Chapter 6 and inventory in Chapter 7. History – the archiving of configuration – is not discussed in a separate chapter as it was not implemented in the Verinec project.

2.3.1 Network Reliability

Network outages can cost a lot of money. If hundreds of employees do not work for one hour, a large portion of the salaries are wasted. If the company looses orders, it can be catastrophic. [Toiga, 1989] for example, confirms that companies suffering an extended period of failure are likely to never recover and go out of business.

According to [Dooley, 2002], network reliability is based on two main principles :

- Fault tolerance. Important parts should be redundant and provide automated failover mechanisms if devices or links are faulty.
- Enough performance and capacity. Even during peak load, the network must still work correctly.

Fault tolerance can be tricky, as the designer must carefully find a balance between just enough redundancy and not too much complexity. Too many redundant network parts not only mean wasted money, they can also introduce new problems. Protocols take longer to stabilise after changes when more devices participate and the added complexity makes it more difficult to understand the network, increasing the risk of mistakes. It would be best to test the network design in order to ensure that it works in an emergency. However, it is usually not wise to test failover mechanisms in a production environment. An NMS should be capable of doing such tests.

Estimating the capacity requirements is important to ensure a reliable performance with a limited budget. It is better to anticipate upcoming network congestion than to wait until the users start complaining. The administrator might find a better way to exploit the existing resources or at least can plan enhancements without emergency pressure.

Several factors are fundamental for a solution that can comply with these principles. The most careful design won't help if mistakes are made implementing it. And performance can decrease dramatically if the network misbehaves or if a hacker gains access. Moreover, an organisational or company network usually does not only have to be reliable but also confidential.

2.3.2 Misconfiguration

Several studies indicate that errors in the configuration of network devices are a major source of service interruptions. [Oppenheimer et al., 2003] for example analysed large Internet sites and found that operator errors are the most important cause of failure. While the Internet sites they examined are quite big, their network structures are not as complicated as the networks of large companies. In company networks, the variety of protocols as well as users at different levels of trust are bound to make operator errors even more likely. Configuration errors are a class of error that could be avoided if the NMS could interpret the intentions of the administrator and assess the network requirements.

[Mahajan et al., 2002] studied the reasons for misconfiguration on border routers running Border Gateway Protocol (BGP). This protocol is used to build the backbone of the Internet. Problems with BGP routers can lead to a loss of connection between parts of the Internet. Mahajan found that a lot of misconfiguration happens: while considering only certain kinds of mistakes, up to 75% of new route announcements are the result of misconfiguration. To overcome this problem, they propose a couple of measures. One is to use high-level languages to directly express the policy and let the software deduce the appropriate low-level setting. Secondly, they propose checking the low-level configuration of the various routers for inconsistencies. The problem of misconfiguration can be further split into several categories:

Syntax: Errors in configuration files that violate the format rules for the service implementation. Depending on the error and the quality of the parser used by the implementation, parts or all of the configuration have to be ignored.

Semantics: Even if the syntax is correct, configuration can still be inconsistent. Settings can be contradictory to each other. This becomes quite difficult to spot when taking the complete network into account, where each device has to be consistent with peer devices.

Requirements: Even if the network is somehow working correctly, this does not necessarily mean that it can accomplish its expected tasks. Policies have to be formulated, declaring which services must be offered, also stating what performance is required. It is only when these policies are complied with that the network is really doing what is expected.

Security: Some mistakes do not directly affect the correctness of the configuration, but do result in security problems. A typical example is a firewall which blocks nothing. Its configuration can be syntactically correct and consistent, the requirements for connectivity are fulfilled, however, it is still not set up in a meaningful way.

While syntax checks are comparatively simple, detecting semantic inconsistencies requires a thorough knowledge of the whole network. Errors in syntax, and to some degree also in semantics, could be avoided if management software would check the configuration. However, most common NMS do not support consistency checks. Standard software often cannot accommodate the specific requirements of companies. Formulating requirements and security constraints must happen on a higher level than on the level of detailed configuration for each service. Security mistakes can only be detected by analysing the configuration, for example with a database of typical errors and best practices. Combining this analysis with the requirements is even better: everything not necessary for fulfilling the requirements should be disabled or blocked. Research is ongoing in this area; the topic is revisited in detail in Chapter 5.

2.4 Network Management Research

Over the years, a multitude of philosophies for network management have evolved. There is no single philosophy which suits all purposes. Home users have different requirements to administrators of a high-reliability network for financial transactions. Management philosophies range from having no concept at all to controlling everything with a central management application and further to self-configuring systems. This section provides a classification of various network management concepts. It identifies network user groups and matches them with the management concepts. A network management system is appropriate for its users only when it minimises the

work necessary to keep the network at the required level of reliability. A balance has to be found between the effort required to learn and prepare a management system and the amount of work the solution will save.

2.4.1 Users of Network Management Systems

The most obvious distinguishing factor between networks is their size. Typical home and small office (SOHO) networks contain a number of personal computers, a few printers and very few servers. They are connected by unmanaged switches or wireless and hopefully protect their Internet access with a firewall. They do not have expert administrators. Usually one of the users with a fairly low level of computer knowledge does the administration. It is acceptable for her to physically attend to each machine. The main criterion is intuitivity. Features should work without explicit configuration operations. Management tools, where needed, have to be self-explanatory. Security depends on prudent defaults and good guidance. If the network is interrupted for some periods, it is acceptable. For example, a faulty switch can be "fixed" by unplugging and re-plugging the power cord.

The larger the network, the less an individual configuration of each machine becomes feasible. Employees are engaged specifically for network administration. Professional administrators can be trained to use more complex management solutions. The main goal is to minimise administration time by automating repetitive tasks, grouping systems for shared modification and avoiding physical visists to the individual machines.

To cope with the complexity, large networks are divided into areas and responsibilities are delegated to various persons or departments. Large companies and service providers can no longer fully trust all of the administrators and need to be prepared to defend their networks from attack by their own employees as well as from external attackers. The network management must provide a sophisticated system of access restriction. The requirements for security in the network are high, forbidding some forms of automated device self-configuration. Switches have to offer port based security features like Medium Access Control (MAC) address limits or Virtual Private Network (VPN) settings and require information on the intentions of the administrators.

Backbones linking large networks and special purpose networks must guarantee a particularly high reliability. Costs for training, tools and extra hardware can be small compared to the costs caused by network outages, as has been explained in Section 2.1. Configuration must be known to work before it is distributed to the network devices.

2.4.2 Network Management Philosophies

Network management should ensure reliability - but costs and effort must be seen in relation to the importance of the network. There is no single solution for all user groups. It is not easy to organise the various features of the philosophies into a hierarchy. Devices that work out of the box are very good for small networks, but in larger networks, security requires thorough configuration. Restrictive defaults are to be preferred. With increasing size, remote administration becomes crucial. Heterogeneity is handled well both by very low level administration and by specialised high level systems, but not with the usual medium-level systems. A feature assuring the correct verification of configurations is missing in most management systems.

No concept

It is natural to have no explicit concept for network administration. This actually works quite well in small networks. SOHO equipment is preconfigured for the most common cases and supports automated self-configuration. Protocols like DHCP and discovery mechanisms provided by Rendezvous for Mac OS or SMB for Windows help to operate networks with no explicit configuration concepts. The security achieved will be moderate at most, but the administration can be done by "amateurs".

The time period between installations of new machines is so long that automating the setup would be a waste of time because new versions of the programs will be available at the next installation. If necessary, new software, such as a printer driver, is installed individually on each

machine. The PCs are often different models and may run different versions of an operating system or even a variety of operating systems.

The network is flat and contains no separation into different zones. There are few changes to the network layout and most devices are unmanaged anyway. If there are server computers, they are administrated by hand. Often the features of the PC operating systems that enable sharing folders and printers suffice, a server dedicated to this purpose is not needed.

Heterogeneity is not much of a problem as long as the administrator can figure out how to handle the various devices. Remote administration is not available, but it is not necessary either. The configuration is not verified and is likely to be inconsistent where it does not directly lead to problems that need to be solved to make the network usable.

There is, however, quite a clear limit to the possible network size when relying on automated self-configuration and non-professional administrators. Nothing can equal a professional administrator who really understands networks when it comes to increasing reliability and security.

Self made

Networks have a tendency to grow from smaller to larger systems. This happens slowly as companies hire more staff and network requirements increase. The know-how of those responsible for the network grows with the network size. At some point, the administrator usually tries to automate the tasks she has to repeat most often. Self-written scripts and various pieces of software for different tasks are assembled.

The administrator herself will know the home-made solution quite well and the solution can work well for the tasks at hand. But the amount of time necessary for developing a good system can become huge. With an increasing network size, the administrator has less and less time for maintaining the management scripts. Very often the documentation of the management system is in-existent or merely rudimentary. When the administrator is replaced or the management team is expanded, it is difficult for new-comers to understand the system.

Deep changes to the infrastructure or changes to the network layout can be another problem. Home-bred scripts are often highly specific and lack the flexibility necessary for adapting to changes in the environment. Adapting to heterogeneity can be difficult, as scripts have to be rewritten. The administrator must figure out for each device how to manipulate it when using scripts; this can be difficult for remote administration. Devices supporting standard protocols like secure shell (ssh) or SNMP have a clear advantage.

Scripting packages

The same problems have to be solved again and again in network management. There are systems available for many system administration tasks.

Two popular examples for this class of management philosophy are LCFG and cfengine. Both were developed within the context of a university network. “Configuration Engine” (cfengine) [Burgess, 1995] has a high-level scripting language and focusses on the file manipulation on Unix systems. “Local ConFiGuration system” (LCFG) [Anderson and Scobie, 2002] focusses on installing software on Unix hosts. Both need agents running on each node. They use a rather simple definition language for the network and are less suited to verifying the correctness of the configuration.

Automated self-configuration of the network becomes less important for many parts, as the scripts control the exact configuration of the network. Remote administration is provided by the scripts and sometimes management daemons on the target systems. Whether heterogeneity is permitted in the network or not depends on the scripting package. If it was developed to support various types of systems or offers enough flexibility to be extended with custom modules, heterogeneity can be handled. Otherwise, if the scripts do not innately support a broad variety of devices and cannot be extended, heterogeneity should be avoided.

Management solutions by device manufacturers

Manufacturers of network devices have an interest in providing the means to configure their devices. Besides the web interfaces often encountered in SOHO products, most of them lanced their own management systems for large installations. Typical examples are the Cisco IP Solution Center or

the Check Point software suite. They make the administration of large groups of similar devices like switches and routers possible.

However, as these management solutions are developed by producers of hardware, they are not intended to operate on devices from different manufacturers.[3] Carriers and other large network companies sometimes decide to radically homogenise their infrastructure to avoid problems with heterogeneous systems. The larger the network, the more expensive such a strategy is. [Mahajan et al., 2002] confirm that "many Internet service provider operators do not use NMS because of the diversity in routers employed. Vendor 'lock-in' is a concern in the industry."

The device manufacturers' management solutions often provide rudimentary configuration tests like syntax checking, but do not verify the correctness of the configuration as a whole before it is applied. They support the proprietary remote administration protocols of the devices and permit the building and administration of large installations.

Verifying configuration

Most network management solutions, even commercial ones, lack one important feature: they cannot check the correctness of the configuration data. Templates are used to avoid mistakes, but the overall correctness of the configuration is not ensured. [Oppenheimer et al., 2003] proposes techniques that lessen failure. The techniques that apply to configuration are *configuration checking* and *reducing the time to repair (TTR) after failure detection*.

It is astonishingly common for administrators to still manipulate network devices individually, using a command line interface (CLI). With direct telnet or ssh access still being the state of the art, the simple introduction of a forced syntax check of configuration would be a first improvement. But correct syntax does not imply that the semantics are correct too. In order to be able to check whether settings are consistent over a network, an overall view of the configuration is required.

One important measure to simplify the verification of configuration correctness is to create an abstraction beyond device-specific configurations. Standardised protocols and services have a limited set of parameters that can be modified. However, the implementations use a wide variety of configuration formats and incompatible syntax, which makes it difficult to test its meaning. [Narain et al., 2003] propose to use abstracted representation of the configuration, oriented towards the parameters of standards rather than the implementation of specific formats. The same idea is used in the Netopeer project [Lhotka and Novotny, 2007], as well as in the Verinec project.

Distributed management

With an increase in network size, the scalability of the management solution becomes an issue. Classical management solutions concentrate all logic in a central management application, which can lead to bottleneck problems. If the machine running the application becomes separated from the network, an administrator has to fix the problems manually before the management system can resume action. Intelligent autonomous agents are more robust when faced with this kind of problem.

[Goldszmidt and Yemini, 1995] propose a concept they call Management by Delegation (MbD). MbD overcomes several key limitations of centralised network management schemes: robustness, scalability and flexibility. Instead of a central logic incorporating every detail of each implementation, an intelligent agent on a device can provide a higher level interface to the management system. The agent produces valid and consistent configuration for each service based on high level instructions. Due to its detailed knowledge of the implementation, it is more capable to handle specific features of the resources.

For a good introduction to distributed network management, see [Martin-Flatin et al., 1999]. They classify distributed management systems by the delegation granularity and the information model employed. Delegation can be centred around network domains, typically around geographic areas, or based on tasks. Tasks can be delegated on a macro or a micro level. For micro-tasks, the delegating agent needs detailed knowledge about the system, and there might be a significant management overhead. Macro-tasks allow the responsible lower level agent to take control and act autonomously, even if there is contact with the agent one level up. The other distinguishing factor is

[3] This restriction improves the autonomy of a manufacturer to harmonise hardware and management software. On the other hand, manufacturers also have an interest in binding their customers...

the semantic richness of the information model. A model based on the managed objects offers only low-level abstraction. A higher level can be achieved by abstracting from computational objects. Basing the model on declaring goals to be achieved provides a very high level of abstraction.

Using intelligent agents and a information model to specify the goals, the NMS could concentrate on modelling the requirements for the network and on defining policies. If a common standard could be found for such agents, this would also solve the problems around heterogeneity.

The Grid and other Autonomous Systems

Large networks are extremely difficult to control from a central point. The concept of *Grid computing* [Foster and Kesselmann, 1998] proposes a radical solution to this problem. In a grid, devices that may have different owners co-operate to provide an infrastructure of computing resources. Computing resources such as storage or CPU power should be available as easily as electricity or the telephone system. [Foster, 2002] defines three key aspects of grids:

- Cooperating resources are administrated by different companies.
- Use open standard general-purpose protocols.
- Able to deliver advanced quality of service.

There are still many open questions. An obvious one concerns security and confidentiality. How can confidential data be stored and manipulated on computers that the owner of the data does not control? There is also the question of acceptance. Will users (companies or organisations) trust the Grid mechanisms to treat their data confidentially whilst not even knowing the geographical location of the actual computers? In the context of Grid computing, human interaction with network management should be avoided whenever possible.

The idea actually is not to have several or various grids, but one single global Grid, similar to the Internet. Standard protocols need to be defined for interaction between grids. These require an understanding of the applications and might have to negotiate on the quality of service, confidentiality and other features.

The increasing complexity of computer networks - whether grids or more traditional interoperating company networks - calls for solutions that minimise human intervention in network management. In [Kephart and Chess, 2003], the concept of a self-managing networks is formulated. They define four self-* properties for autonomic systems:

Self-configuration: Automated configuration according to high-level policies.

Self-optimisation: Systems continually monitor their own behaviour and try to improve performance and efficiency.

Self-healing: Systems detect software and hardware problems and repair those which have only a local impact automatically.

Self-protection: Automatic defences against malicious attacks or cascading failures. Early warning when system-wide failures can be anticipated.

As an idea or a vision, this sounds great. System administrators could focus on planning the network instead of trying to catch up with problem reports. But for the time being, it is no more than an idea, which could take a long time to be realised. The complexity of large and heterogeneous systems does not disappear with self-managing networks, it will simply be concealed from the administrators. According to [Herrmann et al., 2005], several issues have to be solved before self-managing networks can become a reality. They consider the main problems to be:

- Lack of a widely accepted and concise definition of the meaning of self-management.
- Lack of standards for unifying the process of self-management and achieving interoperability in open distributed systems.
- Absence of mechanisms to make self-managing systems adaptive and able to learn.

- Distributed systems tend to be inherently interwoven, which potentially creates an overwhelming complexity.

Other issues, for example concerning security, privacy and trust, also have to be resolved. It might take a long time before all major hardware and software providers agree on common standards and implement them. It will certainly take even longer before parts from different vendors actually inter-operate reliably out of the box.

For self-healing, [Burgess, 1998] defined the term *computer immunology*. During his work on the configuration management system cfengine [Burgess, 1995], he realised how unreliable computer networks are. He proposes a bio-inspired approach to network reliability. Today's computer systems and network management systems flood the administrator with error reports. A system should not only collect statistics about its behaviour, but also understand what is happening and react accordingly. Human interaction should not be requested unless really necessary.

2.4.3 Conclusions for the various philosophies

For small networks, self-configuration is the most important field of research. But as mentioned above, larger networks are less apt for self-configuration both because their complexity makes self-configuration unreliable and because of increased security requirements. The larger the network, the more important sophisticated administration tools are. It is acceptable for administrators of large networks to invest some time in training for a management application.

Intelligent agent-based paradigms can be relevant for larger networks, delegating some of the management from the central system to the network. However, in networks dealing with delicate data, security requires a thorough control of everything that happens in the network. Self-management needs precise high-level policies to avoid security risks. The conclusions of [Herrmann et al., 2005] confirm that it will take some time before self-managing grid systems are ready for productive use. In the meantime, there is still room for innovation within the more classical network management.

The automation of management tasks will become important even before completely autonomic systems can be built. Relieving network administrators of repetitive tasks keeps them free to solve problems which cannot be handled programmatically. In the context of intrusion detection systems, automated reactions to stop attacks on a computer system might permit the defence of a system before it is compromised, or at least reduce damage. The advocates of autonomic systems also agree with the observation that their vision is difficult to realise and suggest a continuous transition towards self-managing systems [Herrmann et al., 2005].

Chapter 3

Network Services

This chapter presupposes a basic knowledge of networks in general. It reiterates some details of services that will be used later on in this thesis. A good overview of computer networks is provided by [Tanenbaum, 2003].

Computers are connected by using various network technologies with the goal of improving collaboration, optimising resource usage and of facilitating access to information systems. Between the physical network (cables, interfaces) and the applications used to achieve the goals of the network, many intermediate devices are needed to produce a smoothly operating network. The various services are arranged in layers, as described by the *Internet model* illustrated in Figure 3.1. The Internet model is similar to the Open Systems Interconnection Reference Model (OSI), but differs in that it combines the session, presentation and application layers, forming one application layer. Physical and the data link layer are referred to as *network access*. The distinction made by the OSI model between data link and physical layer is irrelevant for the Internet model. The Internet protocols TCP, UDP and IP can be run on a number of physical network technologies. Protocols and algorithms have been developed to facilitate the co-operation of the intermediate layers.

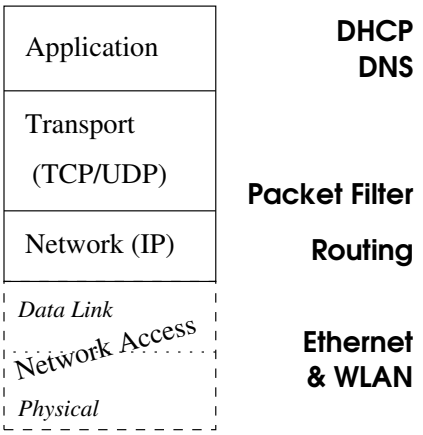


Figure 3.1: Classification of services in the Internet model.

Figure 3.1 shows where the services used in this thesis are located in the Internet model. Two network access methods have been implemented for Verinec, Ethernet and Wireless LANs. Routing is a service of the network layer. Packet filtering, however, crosses the borders between the layers. Besides looking at information from the network layer, it can also consider the data link layer (typically MAC addresses) as well as the transport layer (stateful inspection). Dynamic Host Configuration Protocol (DHCP) and Domain Name System (DNS) belong to the application layer, as they use the transport protocol UDP.

3.1 Network Access Layer

Most Local Area Network (LAN) use one of the Ethernet standards. Ethernet defines a series of protocols to achieve communication over electronic or optical cables. The standard defines different bandwidths, required cable qualities and other details. Bandwidth related information is not taken into consideration here, since performance is not considered in Verinec. One common feature found in every Ethernet standard is the MAC address used by devices to establish communication on the physical network. This unique number identifies the device on OSI model layer 2. As a property of layer 2, the MAC address is used only locally, unlike the global IP address. Network interfaces have a built-in address, but some operating systems permit changing the MAC address. Verinec supports configuring the MAC address. Ethernet negotiates all other connection details automatically, therefore Verinec currently does not support any further tuning of Ethernet parameters.

Wireless Local Area Network (WLAN) is the most common standard for radio communication networks. Because the medium (the air) is open for all devices within range, measures must be taken to select the right network and to protect data. *Access points* are devices used to organise the network communication. Usually they also act as a bridge to the Internet. They are configured with a System Set Identifier (SSID) which is broadcasted to enable the clients to choose a network. As wireless communication is vulnerable to eavesdropping, security measures have been built directly into the WLAN standard. WLANs can operate without encryption or using an encryption standard. At the present time, Wired Equivalent Privacy (WEP) and Wi-Fi Protected Access (WPA) are widely used, although WEP is deprecated as it is contains several vulnerabilities. WEP relies on a single pre-shared key[1] all devices have to know to decrypt the radio communication. WPA eliminated the vulnerabilities inherent to WEP. It can be used with either pre-shared keys or with the Extensible Authentication Protocol (EAP). EAP permits the use of authentication servers like RADIUS in larger network setups.

3.2 TCP / IP

Internet Protocol (IP) and Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) are the base upon which the Internet is built. IP implements the network layer, while TCP provides a connection oriented and UDP a connectionless transport layer. There is only a limited number of things that need to be configured on devices participating in IP. The most important features are the interface address, a 32 bit number, and a mask to determine the network attribution. For readability, IP addresses are usually written as groups of four 8 bit blocks in decimal numbers, for example 134.21.72.77. The IP address can be read as a combination of network and machine number. The network is defined by the left-hand bits of the IP address. The remaining bits specify the machine number. The repartition between the network and the machine part is coded as a mask that filters the IP for the network address. A typical mask is 255.255.255.0, making three-quarters of the IP address the network address part, leaving 255 different addresses for machines in that network.[2] The network information is relevant for routing, as will be discussed in the next section.

Other, more advanced IP parameters usually remain unmodified. These include, for example, changing the maximum packet size to optimise communication with a network access layer that supports a smaller packet size, or modifying the initial Time To Live (TTL) value.

TCP provides virtual connections on top of the IP protocol. It tests connectivity and ensures that the data arrives complete, unmodified and in the correct order at the target application. Usually, TCP requires no configuration at all. In specific situations, some parameters need to be modified. In this way, the number of concurrent TCP connections can be limited, or the "window size" used for retransmissions can be tweaked.

3.3 Routing

Within one particular network segment, all of the communication takes place on layer 2 of the OSI model. Every device involved in the communication has to resolve the layer 2 network address of its partner. In a TCP/IP network over Ethernet, for example, the layer 3 IP number has to be translated into a layer 2 MAC address. The Internet cannot depend on specific layer 2 technologies but needs a generic view on that layer. Different networks implemented with various technologies must be interconnected. Routers, also called gateways, provide two or more network interfaces, each of them supporting a different layer 2 technology. Internally, routers operate on layer 3, the IP layer, which allows them to interconnect networks based on incompatible layer 2 technologies.

Each router between the source and the target network has to ensure that data finds its way from a source network to the target network, where it can be delivered to the target machine. This activity is called *routing*. Packet-based routing, as is used for the Internet, treats each data packet individually. Each router needs some information about the network topology to decide on

[1] Pre-shared key (PSK) means the key must be distributed to all devices by an administrator. There is no dynamic authentication protocol.

[2] The odd 255 comes from the binary nature of IP addresses. $2^8 - 1 = 255$.

which interface to forward packets. However, as there are millions of routers to be found in the Internet, it is not possible for one single router to know the complete topology. This problem is worsened by the fact that the topology is not stable. Links go down or whole new networks are added. Therefore, routing requires decentralised algorithms.

With packet-based routing, the router needs to be able to select on which interface to forward each packet, based on its properties. This decision has to be very fast in order to limit the storage required to buffer packets and keeping the transmission delay low. The decision is taken with the help of a *routing table*. This table tells which networks can be reached via which nearby router over which interface and contains the metrics[3] of each route. The routing table can be configured manually by an administrator or established dynamically using a *routing protocol*. Both possibilities are discussed later in this section. When a packet is received, the routing table is consulted in order to decide which router to forward it to. The result is stored in the forwarding table to avoid recalculating the paths for each packet. The forwarding table describes on which interface to send out a packet, based on the target address and possibly other properties as well. The packet's target address is the most obvious property to take routing decisions on. But some routers can also take into account the type of service, for example treating Voice over IP traffic, which is particularly sensitive to delays, differently from ftp connections. The aim of such policies is to increase network throughput for delay-sensitive applications or to save money by using inexpensive connections when adequate. The routing protocols for the Internet consider only target addresses and the networks they imply. Considering type of service or other factors is left to the individual routers. This is acceptable, as such decisions are specific to the topology of a specific network and its characteristic usage.

Routing for the Internet involves one more level of abstraction. Complete organisational networks are treated as a unit and named Autonomous System (AS). RFC 1930[4] defines an AS as "a group of routers that control a range of IP addresses and follow a common routing policy". Although ASes often belong to one single organisation, this definition allows small organisations to combine their networks into a common AS. Every AS is assigned a unique number which is used for routing between networks. Routers with direct links to routers of different ASes are called border routers. They use an Exterior Gateway Protocol (EGP) to organise the ASes via which traffic is routed to its destination. Although an AS could consist of one single layer 2 network, it is more usual to have routers inside an AS. Internal routers are only responsible for routing traffic within their AS, using an Interior Gateway Protocol (IGP).

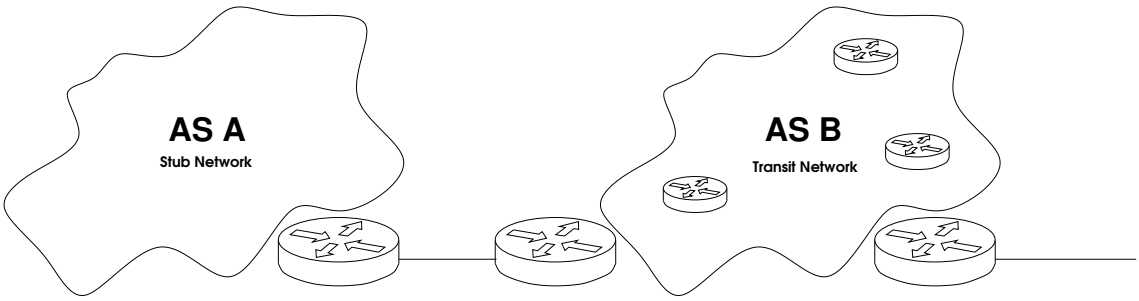


Figure 3.2: Using border routers to connect two networks.

Figure 3.2 illustrates two ASes. The big routers are border routers, while the small ones in network B are interior routers. We will now look at how routing information can be configured on a device and how decisions are made. Section 8.3.1 explains how routing is handled in the Verinec project.

[3]Metrics denote the "cost" for a route. Cost need not be based on money issues, but can also mean distance, bandwidth or other criteria. The information is used to chose the route with the lowest cost.

[4]Standards for the Internet are issued by the Internet Engineering Task Force (IETF). Because the IETF cannot enforce any standard, their propositions are called Request For Comment (RFC).

3.3.1 Static Routing

The simplest routing configuration method is static routing. As the name implies, information about routes is provided by an administrator (or an external application) and remains fixed unless altered manually. Routers with static routing do not adapt to changes in the topology. If the link to a next hop fails, the router continues to send data that will get lost until an administrator modifies the routing table or fixes the link.

Static routing is useful in small networks, as it is simple to configure. An example of a really small network is shown in Figure 3.2. Network A is a stub network, a local network with only one router to connect to the rest of the world. Static configuration bears no risk in such a situation: if the single router is down, there are no alternative routes and even dynamic protocols would not be of any use. Static routing allows a tight control what traffic may be forwarded in the network.

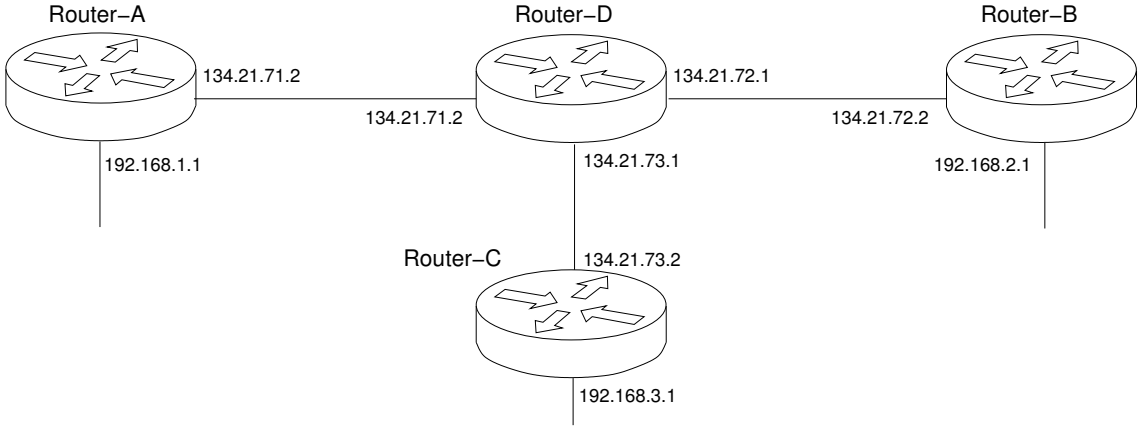


Figure 3.3: A small network suitable for static routing.

Figure 3.3 shows a group of four routers connecting several local networks, each having a network mask of 255.255.255.0. The clients do not have a routing table. Machines in the same subnet can be contacted directly, for all other hosts, the router in the same local network as the sending machine (the *default gateway*) has to be used. The address of the default gateway is either configured into the clients statically as well, or it is provided by DHCP (see later in this chapter). The routing table of router A looks like Listing 1. The administrator constructs the table by listing all subnets in the network. The router needs no gateway for the subnets it is directly connected to. The other networks are reached by forwarding packets to router D. Note that there is no default gateway, as none of the routers connects to an external network. Traffic addressed to networks other than those listed will have to be discarded, and the client notified that the destination is not reachable.

Each route is assigned a metric. The metric is an indication of the "cost" of sending packets over a particular route. In a local network, cost is usually measured in delays caused by intermediate routers and link capacities. For inter-network communication, fees for using external lines can also be modelled in the metric. The higher the metric value, the more expensive it is to take a particular route. In a more complex setup, some destination addresses might be reached by different routes. In that case, the route with the lowest metric value is preferred.

Listing 1: A routing table for router A

```
Destination     Gateway     Genmask         Metric Iface
192.168.1.0     *           255.255.255.0   0      eth0
134.21.71.0     *           255.255.255.0   0      eth1
134.21.72.0     Router-D    255.255.255.0   1      eth1
134.21.73.0     Router-D    255.255.255.0   1      eth1
192.168.2.0     Router-D    255.255.255.0   2      eth1
192.168.3.0     Router-D    255.255.255.0   2      eth1
```

Routers B and C are similar to Router A. The table of Router D is different, as it has three interfaces that connect with all three other routers. Listing 2 shows that table.

Listing 2: A routing table for router D

```
Destination     Gateway     Genmask         Metric Iface
134.21.71.0     *           255.255.255.0   0      eth0
134.21.72.0     *           255.255.255.0   0      eth1
134.21.73.0     *           255.255.255.0   0      eth2
192.168.1.0     Router-A    255.255.255.0   1      eth0
192.168.2.0     Router-B    255.255.255.0   1      eth1
192.168.3.0     Router-C    255.255.255.0   1      eth2
```

A network with more than one router connecting to an external network is also called a *transit network*. In Figure 3.2, AS B is a transit network. It can transport data for the other networks it is connected to, if the border routers are set up accordingly. It is important to correctly configure policies that specify from which source addresses to which target address packets may be forwarded. Otherwise a network might unwittingly transport data for other organisations without receiving compensation. If transit traffic is desired, it is best organised by dynamic routing protocols, as they are more flexible.

3.3.2 Dynamic Routing Protocols

There are two different kinds of routing protocols. An IGP is used between the routers within an AS. Border routers connecting the ASes use an Exterior Gateway Protocol (EGP). EGP and border routers need to be very reliable, as they form the backbone of large networks. Internal routers and border routers also differ regarding how much they can trust peers. Inside the network, a certain measure of trust to the other routers is given, one can expect the system to be oriented towards technically optimal solutions. On the other hand, border routers connect to network parts controlled by other organisations. They must be protected against malicious or badly configured routers. The most important factor for routing decisions on exterior gateways is financial. Different routes can belong to disparate organisations that charge different fees.

Routing Information Protocol (RIP)

Historically, the Routing Information Protocol (RIP) has been very important. Its underlying algorithm was already used in the first version of ARPANET. The RIP algorithm tries to establish the number of hops (intermediate routers) needed to reach the target network. This classifies RIP as a *distance vector* routing protocol. The metric of a route is simply its hop count. Periodically, every router participating in RIP has to broadcast its full table of routes to all other routers.

RIP has become obsolete due to its various limitations. The maximal distance between any two routers can be no more than 15 hops. The value of 16 is used to indicate that there is no route to a particular network. This number was certainly adequate in the early days of ARPANET, but is often too small in current large networks. The limitation is difficult to overcome, as RIP relies on counting up to 16 to mark unreachable routes as unavailable. Even as it is, RIP can be quite slow in converging after a change.

This is best illustrated by an example. Let us assume that the network in Figure 3.3 runs RIP. In its table broadcast, router D will advertise that it has a route to B with costs of 0. A will store this route. When using standard RIP, A will advertise this route to D in the next update cycle. Because the costs for the route to B over A are higher than its direct route, D discards that information. But let us now assume that router B fails. In the next update cycle, router D receives the route information from A, stating that it has a route to B with a cost of 1. Router D will happily add this information to its table (with costs of 2 for the additional hop) and send packets for B to A. A knows from its table that D has a route to B and sends the packets back, resulting in a tight loop until the TTL for the packets is reduced to 0. With each update, A and D send out their route with the metric increased by one, until they reach 16, at which point they realise that B is no longer reachable. Until then, data loops between A and D. This problem known as

"count to infinity" has been solved using a scheme called *split horizon*: the hosts do not advertise the routes on the interface they came from.

There are more problems to RIP. Loops between more than two routers can occur even when split horizon is used. The full updates can produce a lot of overhead in a slow network or in a network with a large number of routers. Yet another problem is that older protocol versions did not provide authentication. A malicious computer acting as a RIP router could mess up the complete network. Because of these limitations, other protocols have been developed and have replaced RIP in most installations.

RIP has a number of settings and options. A group of co-operating routers use a common network-number. This makes it possible to have more than one group of routers operating on the same network. The split horizon feature can be enabled or disabled for certain network types. The timing can be set to tune the behaviour of the network:

- how often routing updates are sent.
- after what period of no connectivity a route is declared invalid.
- how long information regarding better paths is withhold, in order to facilitate stabilisation.
- how long normal routing updates are delayed.

RIP version 2 also uses an authentication concept that excludes malicious or broken routers from the protocol. For authentication, a key-chain and the mode, which is either plain text or md5, have to be set.

Open Shortest Path First (OSPF)

One of the commonly used routing protocols in TCP/IP networks is Open Shortest Path First (OSPF). It is a *link state* protocol - each router builds a graph showing the routers it has a connection to. To keep the graph at a reasonable size, the network is divided into areas. Routers only need to know the graphs of areas they are participating in. For routing decisions, the connectivity graph is transformed by using Dijkstra's algorithm [Dijkstra, 1959]. The result is a tree with the router running the algorithm as root node. The shortest path to any router is the list of nodes that must be traversed in the tree to get to the node in question. Whenever the link state changes, the graph is modified and the tree adapted to the new situation.

The map of an area is established as follows. Every router determines what other routers it can directly reach via its interfaces. It assembles that information into a link-state advertisement. One router is elected the Designated Router (DR), which receives all link-state advertisements and builds and broadcasts the connection graph. For the connection graph, only connections advertised by routers on both ends of a link are taken into account. If only one end advertises a connection, the link is considered broken.

The basic link state protocol would have the advertisements broadcasted and requires every router to build its own table. Electing and using a DR for link state aggregation is an optimisation in OSPF to reduce broadcast traffic.

OSPF requires some configuration on the routers. Every router in OSPF needs a unique ID. The router ID has the same format as an IP address. The network interfaces should be assigned to OSPF routing areas. For each interface, costs can be defined for the routes using it. It is also desirable to be able to define costs for specific neighbours, as they are not necessarily all connected with equal bandwidths. Various timing and authentication parameters can be set, similar to RIP. Each area can be set to use broadcast, non-broadcast or point-to-multipoint for route distribution.

Border Gateway Protocol (BGP)

While the previously discussed protocols are both IGPs, Border Gateway Protocol (BGP) is an EGP. Historically, other EGPs did exist, but today BGP is *the* protocol used to route between ASes in the Internet.

BGP has some similarities with distance vector protocols such as RIP. However, instead of simply distributing routes with metrics, BGP routers include the path a packet taking that route would traverse. Therefore the protocol is classified amongst the *path vector* algorithms. The path

is expressed as a list of the AS numbers on the path. Routing loops are reliably detected, as the routers check the path and discard any routes already including their own AS.

Neighbouring routers connect to each other using a TCP session and periodically check whether the peers are still alive. If a link fails, they select a replacement route - if there is one left - for networks that were previously reached over the failed router. When the routes change, the router must announce an UPDATE message to its peers. If the update influences their routes, they need to further distribute the new routes they are using.

Routing decisions in BGP are not made according to the technically best solution (the fastest connection) but based on economic criteria. The routers are configured by policies describing whether they forward traffic to certain destinations or not. The distance is measured only by the number of ASes to traverse. The size of an AS can vary a lot, but is not taken into consideration by BGP. The same holds true for the bandwidths of links, which are also ignored. As each router announces only the best route to its peers, load balancing is not possible. Announcing alternative routes could also speed up recovery when a change has to be made. On the other hand, the size of the routing table is already a problem without the additional information of alternative routes. Every router has to have information on all routes. Although link aggregation is used to merge paths based on the same IP address prefix, an Internet BGP router currently has to know about 200'000 routes, and the number is growing fast.

For BGP configuration, there are a couple of settings that must be considered. Of course, each router needs to know which AS it belongs to. Because BGP is a backbone protocol, it would not be acceptable to have routers spontaneously setting up links to routers they have detected. Thus, information about neighbouring routers and routers belonging to the same AS is provided manually. Parameters to tune route distribution and weighting are implementation-specific.

BGP is not restricted to inter-AS communication. Very large networks can also make use of a BGP to connect several network areas, which might run OSPF or another protocol internally.

3.4 Packet Filtering

A network has to be protected from unwanted traffic and there should be limits set for the kind of data leaving it. In analogy to buildings, where special walls are built to prevent a fire from spreading unhindered, the term *firewall* is also used for means of avoiding uncontrolled data flows. Both the software doing the filtering as well as the concept of rules defining what data should be filtered are sometimes called firewalls. To add to the confusion, there exists dedicated hardware for protecting the network. Such devices are sometimes called hardware firewalls, while firewall applications running on general purpose computers are called software firewalls. Dedicated hardware is more secure because no other service runs on it, reducing the potential vulnerabilities. However, the distinction between hardware and software firewalls is not important from the conceptual point of view. The actual filtering is always done by some kind of software, and this software always runs on hardware. In this thesis, 'firewall' denotes a device that filters network traffic, regardless whether it is a dedicated device or not.

Firewalls are usually placed at the border between two different zones of trust. Any extraneous traffic is an unnecessary risk, because it could trigger harmful effects. Attackers could glimpse information about the network to plan their strategy. Forged packets can sometimes cause applications to break or even open a hole by which the system can be entered. Outgoing traffic is limited to control which stations may have access to the Internet and to limit the damage in the case of a successful intrusion. The aim of a firewall is to limit as much of the traffic as possible while still keeping the network usable. Traffic is usually controlled by looking at the packets passing the firewall and deciding whether to forward or drop them, hence the name *packet filtering*. This can happen on different layers of the network. On the network layer, individual packets can be controlled, based on IP header properties like addresses, ports or flags. Information from adjacent layers can also be taken into account. The data link layer provides Ethernet MAC addresses. In the transport layer, information about the state of connections is used to decide whether a packet is expected or not. This is called *stateful inspection*, because the firewall looks at the state of a TCP connection, or logs information about a UDP communication. As the packets are not assembled in packet filtering, the payload cannot be checked.

On the application layer, proxies can check the contents of a protocol. For example, access

to certain web pages can be restricted, or emails containing certain keywords can be filtered out. Application layer firewalls do not forward packets, but assemble messages. If the rules allow a client to perform a given action, the proxy opens new connections on its own, forwards requests and sends the responses back. Since proxies must analyse the contents of the communication, they need to understand the protocol. There is usually a separate application for each protocol that must be support. The remainder of this section will focus on packet filtering and not discuss application proxies.

Some packet filters run on the same machine they are supposed to protect. They filter only the traffic between the local machine and the network. This concept is usually called *personal firewall* [Cheswick et al., 2003]. The advantage of personal firewalls is the possibility of direct interaction with the user and the option of controlling which application initiates network connections, thus allowing for more precise filtering. The drawback is that if the personal firewall is flawed, an attacker directly gains access to the protected machine, providing less security than a separate firewall device. This kind of firewall is typically configured in a dynamic process between the user and the firewall application. The Verinec project does not address personal firewalls specifically. A classical firewall concept is assumed: a separate machine with (at least) two interfaces decides for each packet whether to forward it or not. Filters for traffic addressed to the local machine or originating from the machine can also be specified.

3.4.1 Netfilter

In the Linux world, the most common packet filter is the Netfilter framework [Welte et al., 2006]. It has been part of the Linux kernel since version 2.4. The visible front end of it is the command line program `iptables`. This framework uses a concise concept to handle the different flows of packets and thus exemplifies the operation of a firewall. Besides packet filtering, Netfilter is also capable of manipulating packets, a feature called *mangling*, for example to do Network Address Translation (NAT). This section is focused solely on packet filtering.

The Netfilter framework handles packets in different *tables*. There is a table for filtering, called the 'filter' table. The other predefined ones are 'nat' and 'mangle'. Every table contains *chains* grouping individual *rules*. The table decides for each packet which chain to use. In the filter table, there are three built-in chains, providing the entry points for processing. They are named:

INPUT: Packets addressed to this machine.

OUTPUT: Packets sent from this machine to the network.

FORWARD: Packets received from the network, but addressed to some other machine.

It is possible to add custom chains which can be called upon match of a rule instead of simply applying one of the default actions. Custom chains allow isolating a sequence of rules to be used from different chains. Together with the RETURN action, custom chains allow the implementation of a kind of function call mechanism for packet filtering.

By default a chain is empty and its policy is to let all packets pass. The chains need to be filled with an ordered list of rules before they actually do anything. When a packet needs to pass a chain, it is checked against each rule in that table. A rule specifies tests and a *target* to use if the test is successful. Typical tests are destination or source IP, receiving network interface or whether properties like bits in the header are set or not.

Each rule consists of the following elements:

target: Target or chain to jump to if rule criteria match.

prot: Of which protocol the packet has to be to match the rule.

opt: Options for the packet. For example whether it is fragmented or not.

in: Interface the packet has to be received by.

out: Interface the packet is going to be sent from.

source: Source IP address of the packet.

destination: Target IP address of the packet.

options: Additional criteria, depending on the protocol specified in prot.

A *target* can be either the name of a user-defined chain or one of the default targets. The default targets are used to decide on the packet's fate. Default targets are:

ACCEPT: The packet is accepted, no further rules in this table need to be checked.

DROP: Silently drop the packet immediately, do not send any notification to the sender.

REJECT*: Drop the packet and send specified message back to the sender.

RETURN: User defined chain: return to the calling chain. Built-in chain: execute default target.

LOG:* This target does not stop the chain or decide anything. It is only used for its side-effect of writing a message into the log file for debugging or auditing purposes.

Targets with a star (*) are implemented as extensions in Netfilter. Other more advanced targets are QUEUE to pass the packet to a user space program that can decide what to do with the packet, ULOG for safer logging and the NAT targets DNAT, SNAT and MASQUERADE.

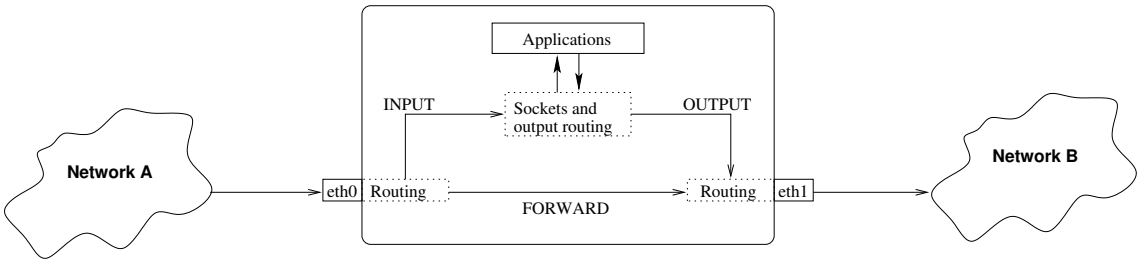


Figure 3.4: Flow of packets in the Netfilter framework.

The flow of packets is depicted in Figure 3.4. For each packet arriving on an interface, Netfilter first has to decide whether the packet is destinated to the local machine or should be forwarded. Depending on that decision, different chains are used. The pre- and post-routing chains used in nat and mangle are not shown in the figure. They are located between the physical interface and the routing decision.

Section 7.2.3 discusses how packet filter configuration can be imported into a configuration database on the example of the `iptables` command. Section 8.3.2 explains how packet filtering is handled in the Verinec project.

3.5 Domain Name System

The Domain Name System (DNS) is used to look up IP addresses for host names in a network. On IP level, all packets must bear the addresses of the source and target machine. The addresses are coded as numbers. For example, 198.162.0.1 is an IP version 4 address. This naming scheme should be hidden on the application layer to ensure a clean design – and human users will be grateful not having to remember long arbitrary numbers. To allow for names independent of the IP layer, DNS has been developed. This protocol allows the resolution of humanly readable names like `www.unifr.ch` into a number suitable for the Internet Protocol (IP). There is also *reverse DNS* to convert numerical IP addresses into a domain names, which is mostly used for logging and debugging. A good introduction to DNS can be found in [Tanenbaum, 2003].

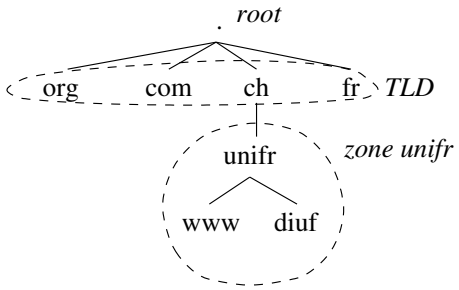


Figure 3.5: A small excerpt of the DNS tree for www.unifr.ch.

The DNS system is built hierarchically, as illustrated in Figure 3.5. Domain names are read from right to left. The rightmost part is the top level domain (TLD), usually a two letter country code or a generic code for a class of organisations. Each name left of the top level domain, separated by a point, designates a subdomain. There is a list of root servers with known IP addresses to indicate the servers for top level domains. The responsibility to resolve names can be delegated to a different name server for each subdomain. The area of responsibility of a DNS server is called *zone*. A zone encapsulates configuration data for a domain and its subdomains. It can also indicate that other name servers are responsible for certain subdomains. It is, however, not necessary to have a separate DNS server for each domain. One server can control more than one zone.

DNS is a client-server based protocol. The *clients* do not need much configuration, only the IP addresses of one or more servers to contact. This information is interface-specific, either provided by DHCP or configured statically on the network interface. A DNS *server* does two things. It provides information about its zones, which must be available in a configuration database. For hosts outside of its own domain, it relays the request to the responsible DNS server and then caches the answers for further use. Looking up and caching do not require any special configuration. What matters is the configuration of the domains controlled by the institution. The basic DNS entries are name to IP mappings (addresses, A), reverse mappings of IP address to host name (pointer, PTR), alias names (canonical name, CNAME) and mail servers for addresses @ that domain (mail exchange, MX).

Section 7.2.4 discusses importing configuration files from the BIND implementation. Section 8.3.3 explains how DNS configuration is handled in the Verinec project and Section 8.5.5 provides details about the translators for the DNS servers BIND and djbdns.

3.6 Dynamic Host Configuration Protocol

The Dynamic Host Configuration Protocol (DHCP) is used to assign IP addresses, host names, netmasks, default gateways and other IP configuration to client machines at startup. Thanks to this protocol, machines can just be connected to the network and obtain a usable configuration automatically. The client configuration is quite simple, it just has to be told whether it must issue a DHCP request or use a static configuration. Configuration on the server side is more complex. At the very least, the server must be given a range of IP addresses it is allowed to assign to clients, as well as the other IP parameters. Most servers also allow rules to further define how IP addresses are assigned. One option is to define a fixed mapping from MAC addresses to IP address. If a machine is not in the list of known MAC addresses, the server can be told to assign it an IP from another block, or to exclude it from the network by not issuing an IP for it.

The DHCP protocol is connectionless. A client sends a *DHCP discovery*, which is broadcasted, since the client does not have any information on the network it is in. The discovery message tells the MAC address of the client machine to let the server know where to send its answer to. DHCP servers listen for such broadcasts and respond with a *DHCP offer* message providing the necessary configuration data. After a successful IP assignment, either the client or the server should inform the DNS server about the new hostname - IP pair. Some DHCP servers allow the configuration of this feature, although DHCP clients of all modern Operating Systems can inform the DNS servers by themselves.

If there are several DHCP servers, which is recommended since it increases redundancy in case one of them crashes, there might be more than one DHCP offer sent in response to a discovery broadcast. The client has to choose which offer to take and broadcasts a *DHCP request* for the selected offer. The server with the selected offer responds with an acknowledgement. The other DHCP servers see the request and treat it as refusal of their offer. If they had reserved an IP for the offer, it is released at this moment. Redundant DHCP servers should be configured to be responsible for distinct blocks of IP addresses, in order to avoid having the same IP assigned to

two different machines.

The server response also sets the 'lease time', i.e. the time until the information expires and an update should be requested by the client. The server marks the IP it has assigned as used until the lease time is finished. If the client is still running, it can send a request to keep that address.

The client can ask for a specific IP address by specifying it in the request, usually an IP it has previously been assigned. If the address is still available, or reserved for that client, the server will accept the request. However, the address might have been assigned to another machine in the mean time, or come from a different network and would be invalid in the current network. If one of the DHCP servers is responsible for a requested but unavailable IP, it sends a *not acknowledged* message, causing the client to ask for a new address. But when a machine is moved between two different networks, as happens with laptops, for example, there may be no server responsible for an address. Then the client waits for a response to its request until timeout, which can take several minutes. At least one of the servers should be configured to be authoritative, meaning it will send a not acknowledged message for any address not in the network. In a setup with redundant DHCP servers covering different IP blocks, the authoritative server of course needs to know about all of the blocks, or it might refuse valid requests.

Section 8.3.4 explains how DHCP configuration is handled in the Verinec project.

Part II

Network Configuration

Chapter 4

Use Cases

To get an idea of the different tasks a network management solution should support, this chapter presents a series of use cases. It is focused on tasks covered by Verinec, although there are many other challenges in network management.

4.1 Configuring a Complex Network Consistently

Devices involved in the same communication should be set up in a consistent manner. While it is relatively easy to check whether one device is configured with values according to the specifications, it is more difficult to determine whether all of the devices are set up in a way that allows them to communicate with each other.

For example, take the case of Figure 4.1. It is a network in a two-floor building. On each floor, there is a subnet with client machines. Each subnet has a gateway router connecting it to the server. If one of the clients wants to connect to the server, it must use the address of the gateway it is actually connected to. It is ineffective if the client tries to use the gateway of the subnet of the other floor. Should client1 try to use gw-floor2 as its gateway instead of gw-floor1, the client will not be able to contact to it, in spite of the fact that the gateway does exist in the network. A simple search for the gateway in a configuration database of listed machines present in the network is not enough to show that this setting will not work. The network layout has to be taken into account to detect inconsistencies of this kind. Thus, in order to assess the correctness of the configuration, the management application needs detailed knowledge of the network layout, as well as the devices in it.

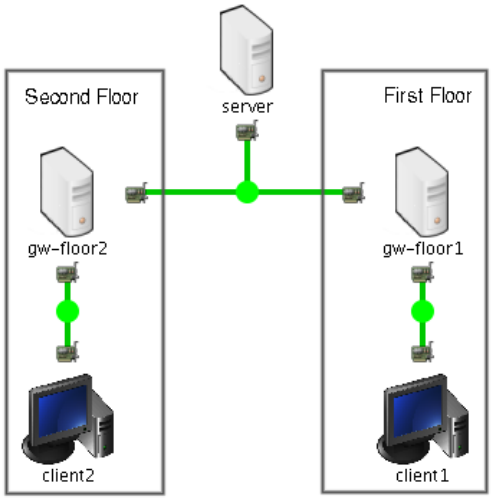


Figure 4.1: Consistent network.

4.2 Improving the Configuration of an Existing Network

Frequently, a network grows over time. In the beginning, it may be small enough to be administrated manually. Later on, a home-bred solution with scripting or professional administration tools for certain aspects of management is employed. Gradually, the network becomes too complex to allow for an overview without sophisticated tools. Configuration settings become inconsistent because changes in the topology are not reflected in the setup of all of the devices. Since problems are usually fixed under pressure, the administrator leaves things be as soon as they work, no matter how. Temporary workarounds for problems can remain in the firewall configuration, creating security holes. At some point, the network requires a thorough clean-up. One solution could be throwing all existing configuration over board and starting again from scratch. Designing the network from scratch is possible, however, it means redoing a lot of already existing correct configuration. Even if a network is not optimal, there still is a lot of usable information to be gained from it.

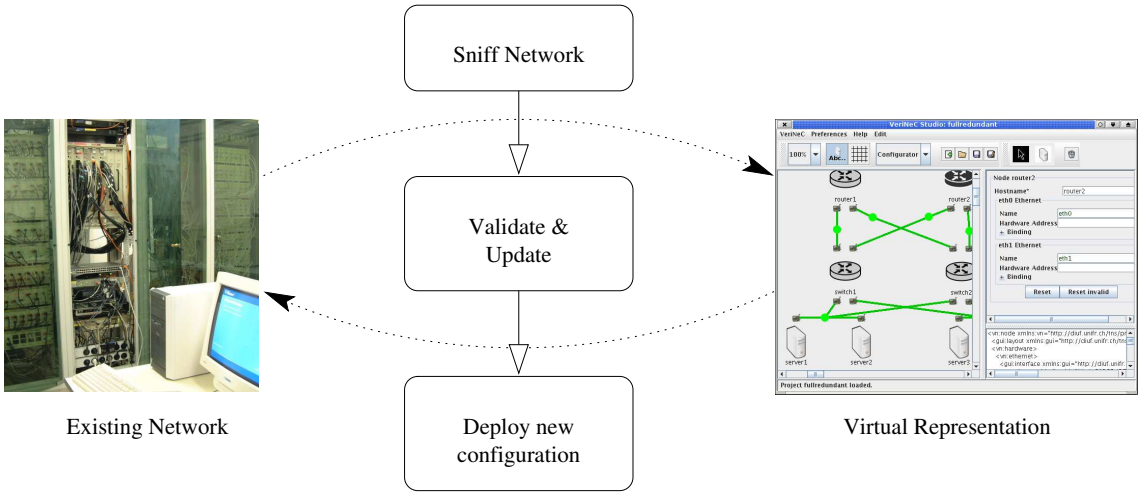


Figure 4.2: Improve an existing network.

Figure 4.2 illustrates the process of using an existing configuration while improving the network. Analysing the traffic within a network provides indications about the hosts that are in it. This information can be completed by traceroute in order to find the routers and port scans to gain more knowledge about the hosts. In a second phase, configuration data can be imported into the management solution. When all configuration is known, the management solution can examine it and run tests on it. The software should aid the administrator in determining problems and in fixing them. Tests can be used to check basic validity and congruence. Network simulation is used to test complex requirements. Once the configuration has been improved, the management solution will automatically reapply it to the network devices. If cable connections where modified within the management solution, the administrator will also have to apply these changes in the real world.

4.3 Configuration Depending on the Environment

Templates are used to reduce redundancy in configuration. The template defines configurations that can be re-used for a certain task, marking the places where specific information must be added. When the template is applied to a particular machine, parameters must be provided. Many of the parameters follow from the surroundings of the machine, some in a physical sense like the location, others in a logical sense like the IP subnet. The printer to use is the one in the same office, the default gateway is a router specific to the subnet of the machine and so on.

Instead of entering such values by hand, they can be automatically acquired from the environment. [Gil and Lorenz, 1996] uses the concept of *Environmental Acquisition* to solve problems of this kind with programming languages. In [Buchmann et al., 2006], the author of this thesis discusses how to adapt this concept to network configuration. The configuration template indicates which values should be acquired from the environment. The network model specifies hierarchies of environments and contains the definitions for the variables. The values are updated when either the setup of the environment or the attribution of the machine to a network (the location) changes. Environmental acquisition is convenient and intuitive for the user of the configuration system, as the settings are updated as soon as the location of a node is changed. It also has the effect of reducing redundancy, which helps avoid inconsistencies. Figure 4.3 illustrates a situation occurring when a machine is moved from one room into an other. In the management solution, this change implies changing the printer name. The machine is then updated with the new configuration and ready to operate.

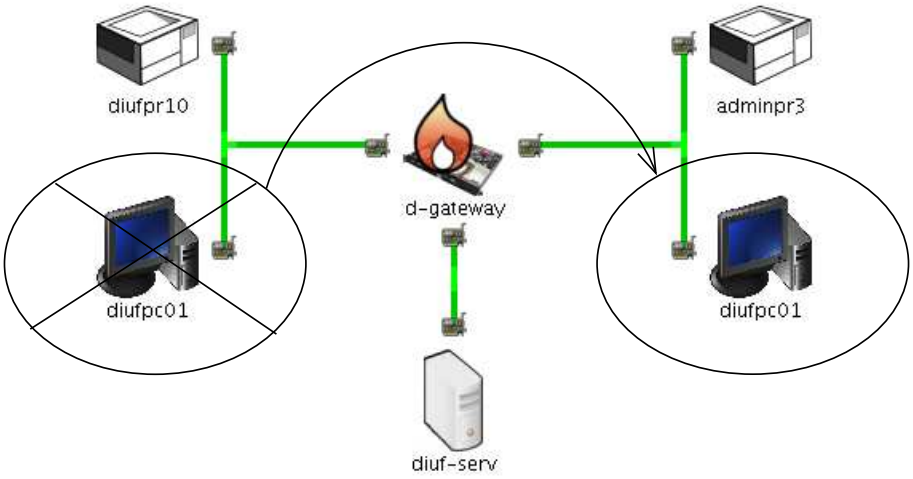


Figure 4.3: Retrieve settings from environment.

As a side note, consider dynamic configuration protocols (typically DHCP). Usually, a machine configured to participate in automated configuration processes broadcasts a request into its current *environment*. A server responds with information on how to configure the machine for the current network. This can be seen as environmental acquisition in a very literal way. As these protocols configure the devices at run-time, dynamic configuration does not require the NMS to know about the environmental acquisition. Nonetheless, it still makes sense to take the concept into consideration in an NMS as well. Dynamic configuration protocols are not adequate for all settings and can lead to security risks. In larger networks, stability is more important than operating "out of the box". When static configuration is preferred, environmental acquisition can emulate the functionality of DHCP at configure time.

4.4 Replace Broken Component

When a device is broken, the administrator needs to replace it as soon as possible. This section shows two cases of replacement. In the more simple case, which can be handled with any decent backup strategy of configuration data, the administrator has an identical spare device that can replace the defective one. Things become more interesting if, for some reason, the only available device stems from a different manufacturer. Here the Verinec approach to configuration management can demonstrate its full strength.

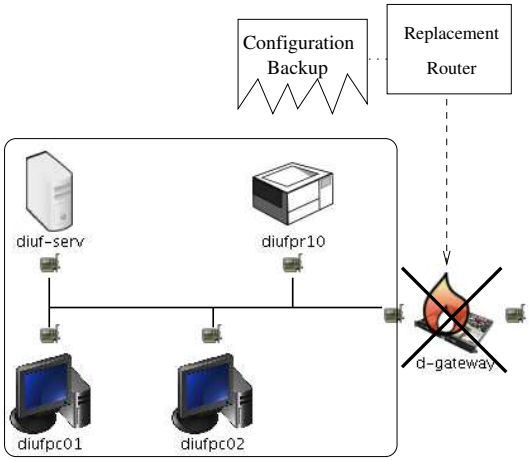


Figure 4.4: Replacing a device with another of the same type.

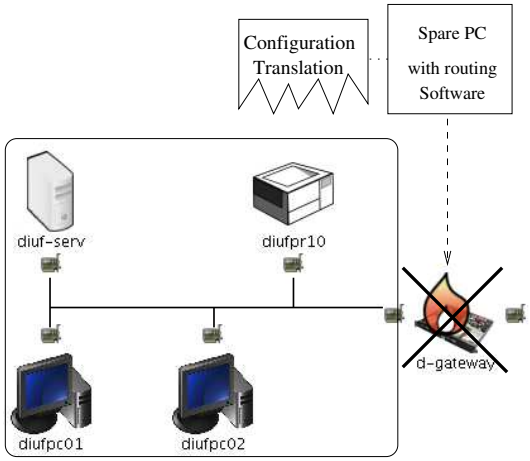


Figure 4.5: Replacing a device with one from a different manufacturer.

4.4.1 Replacing a Device with one of the same Type

When a network device does not operate any more, the network administrator has to replace it. If the configuring was done directly on the machine and there is no backup, all of the configuration is lost and must be re-established.

To avoid this, there should at least be a backup of the device configuration that is stored in such a way that administrators can find it and apply it to the new device. If the configuration was always modified outside of the device and uploaded afterwards, the file will provide a backup of the configuration that is up to date. Figure 4.4 illustrates the situation.

4.4.2 Replacing a Device with a Different Product

Sometimes there is no replacement device of the same model available. In this case, the administrator must use a different model, perhaps even from a different manufacturer, to quickly re-enable network operations. For example, when a firewall appliance is broken, she could take a PC with Linux on it and configure iptables as a firewall, until a replacement device arrives. The configuration backup of the firewall appliance might aid her in knowing what rules to apply, but she has to study them and configure the replacement device by hand. If the configuration is stored in a device-independent format and translated automatically, it is as simple to use a machine from a different manufacturer as it is to employ one of the same type. This is illustrated in Figure 4.5.

This feature can be useful even when the original device is still in working order. Almost the same problem arises when an older device has to be replaced with a newer one that is more powerful or more reliable. This use case applies to emergency replacement as well as to planned upgrades of the infrastructure.

4.5 Roles of computers in a network

Consider the case of a small company that has 20 workstations, some servers and an Internet access. An intranet router separates the workstations from the file and web server. The servers are connected to the Internet through another router. Those servers are in an isolated area, called Demilitarised Zone (DMZ) [Dooley, 2002]. Figure 4.6 shows the setup.

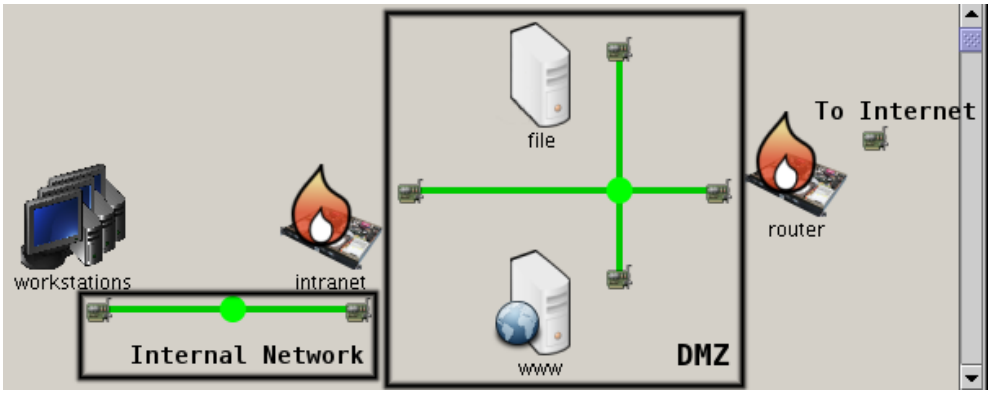


Figure 4.6: Use case network in the Verinec editor.

The network administrator formulates a couple of policies for the different types of devices in the network. Certain actions must take place, while other policies forbid certain operations. The rules are depicted in Figure 4.7. Requirements are denoted with a + sign, restrictions with a - and narrowed requirements with a >.

- The workstations must be able to initiate connections to the Internet only for the HTTP protocol and must not be reachable from the outside.
- The file server must neither be able to reach the Internet nor be reachable from outside, whereas workstations must be able to reach the file server.
- The web server must be accessible from the Internet and by the workstations and must be allowed to connect to other hosts on the Internet. However, it must not be able to initiate connections to the workstations.

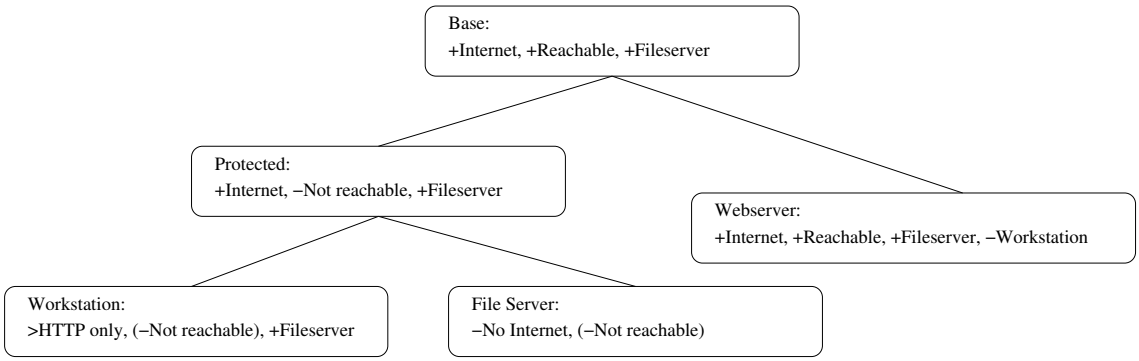


Figure 4.7: Role model for a small network. Automatically inherited restrictions in parenthesis.

Now the administrator is informed that some users need more Internet access than just the World Wide Web for their work. To avoid allowing all users full Internet access, she can split the role given to the workstation into “Safe Workstation” and “Power Workstation”. The Power Workstations now inherit the full Internet access permission from the “Protected” role, while the Safe Workstations have the same restrictions as the “Workstation” before.

4.6 Determine Relevance of Configuration Errors

In large networks, the network configuration must realistically be expected to have inconsistencies. Problems have to be solved according to their impact. However, deciding which services depend on which other services requires a thorough knowledge of the network. Dependencies can be quite hidden and cross all network layers.

Figure 4.8 illustrates the problem with an example. The machine *client* makes a request to the web server *server*. The client uses a public DNS system to resolve the IP of the server. The server in turn has to access *fileserver* to fetch the files requested by the client. There is a firewall between the web server and the internal network where the file server is situated, but it contains a rule that allows access from the web server to the file server. However, the network administrator forgot that the internal DNS server is protected by the firewall. The public DNS knows nothing about the internal file server and thus the web server can not access the files although the firewall does allow the server to connect to the file server.

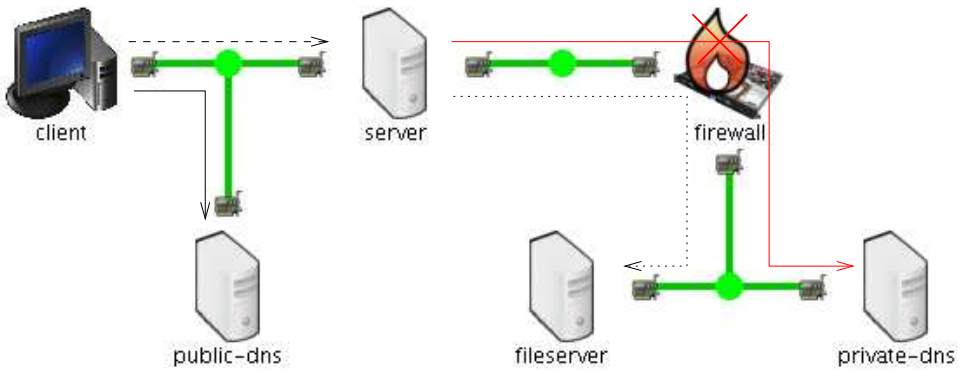


Figure 4.8: Complex dependencies: the request fails because of indirect problems.

If the administrator validates the configuration of each service separately, she sees only a problem that concerns the web server reaching private-dns, but may think that the connection to the fileserver should work since it is allowed. Although the connection between client and web server is set up correctly, the system as a whole will not be able to provide the expected service.

Now why does the administrator not simply fix the problem with the DNS? Firstly, she might not even consider it an issue, since she thinks it is not necessary. Even if the block is reported, there might be lots of similar warnings in a large network. Without further aid, the administrator has no way of determining which warning concerns something important that should be fixed with priority. The administrator should see at one glance what major services fail because of a problem. A simulation of the network can provide this information without requiring the user providing details of the dependencies between services, as long as the configuration of all services is available.

4.7 Redundant Failover Capabilities of a Network

A reliable network needs to be operable even if some components stop working. Redundant hardware is the base upon which specific algorithms and fallback concepts are built. To adapt the network in the case of device failures, the Spanning Tree algorithm on MAC layer and dynamic routing algorithms on the IP layer are used. However, whether the provisions to handle a malfunctioning device work as intended only becomes visible when a failure occurs and it is too late to fix any errors. The first step is trying to identify "single points of failure", that is, a device that has no alternatives. If the single point of failure breaks, this interrupts network operations. Single points of failure can be eliminated by introducing redundancy: important devices and links are duplicated.

An example of a single point of failure is shown in Figure 4.9. There is one switch connecting the servers with each other and with a router that provides a connection to the Internet backbone. The network could be a simplified web hoster with a couple of server computers that have to be connected to the Internet. It is important that they should be reachable from the Internet all of the time. They are located in two computer rooms and connected by a switch.

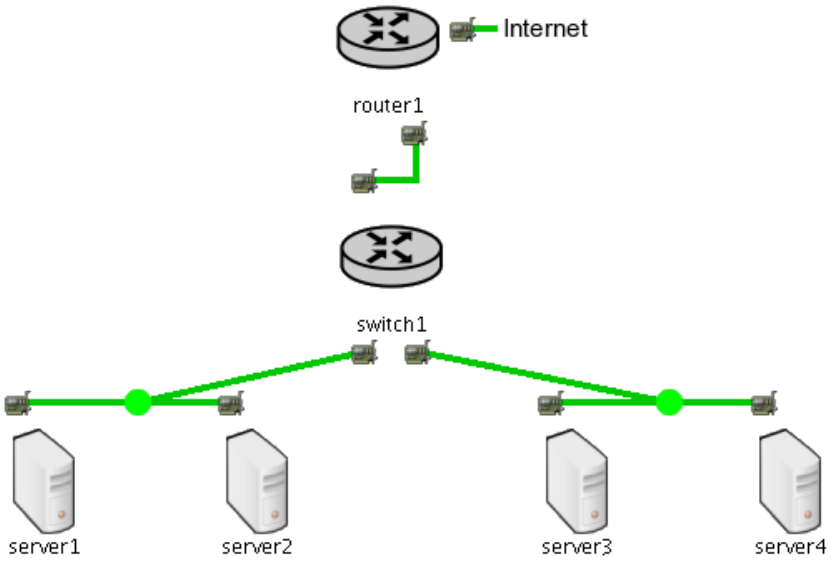


Figure 4.9: Network without redundancy

We can see clearly that this layout has two single points of failure: if either the router or the switch breaks, the servers are cut off the Internet. In a complex, real-world network, single points of failure are not always that easy to spot. It would be useful if the NMS could identify them automatically and even more useful if it could also indicate the relative importance of that point. Not all devices are equally important and a network with complete redundancy is neither sensible nor affordable.

If a simulation of the network is possible for one configuration, it is also possible for a series of configurations, each with one of the devices removed. Whenever something important stops working in a simulation, the device that was removed for this simulation run is a single point of failure. As in the previous section, the importance of a single point of failure can only be determined by an administrator looking at the result. If many failures are to be expected, a strategy that determines the importance of each will be necessary.

Chapter 5

Verification of Configuration

Section 2.3.2 discussed the different types of mistakes that can happen in configuration: Syntax, Consistency, Requirements and Security. Correct syntax is a fundamental requirement for the network devices to run. However, many settings depend on each other if a network is to operate as expected and these cannot be expressed in the syntax. Even if the configuration is consistent, this does not automatically mean that the requirements for network usage and security constraints are fulfilled.

Several methods can be used to verify the correctness of the configuration in respect to these factors. This chapter discusses the concepts of semantic rules, logical reasoning and simulation. There are rules that check static properties and that can be used to define the requirements of some configuration settings. However, some requirements cannot be checked by static rules. Two methods can be used to extend the verification further. One method consists of simulating the network and analysing the result in order to find out whether the required operations really occurred during the simulation. The other possibility is to employ logic reasoning systems to verify the correct configuration of each system that provides a service. The latter will be discussed in Section 5.7 under related work.

The steps that verify the correctness of configuration data have a lot in common with translating program source code into a correct executable file. Before looking at configuration verification in detail, the following section compares code and configuration.

5.1 Analogies between Configuration and Program Code

Regarding correctness analysis, configuration data has a lot in common with program code. Program code is written to control the operation of a (possibly virtual) computer system. Configuration in turn is written to control the software defined by program code. Programming languages are classified from low level (machine code) to high level. Since each level is more abstracted from the underlying hardware, the expressiveness of the languages increases with the level of abstraction. Usually, the configuration data is easier to understand than a programming language. It is specifically targeted at the domain of the particular program to configure.[1] Configuration can be seen as a higher level programming language for specific tasks. It is abstracted from the programming language used to implement the functions.

From this point of view, many similarities between the two are revealed. Both have syntactic and semantic properties and result in a certain runtime behaviour. But while testing program code is common practice, the verification of configuration is less usual. One explanation might be that configuration data is often smaller and simpler to understand than large programs. However, if the complexity of a complete network is taken into account, any idea of simplicity clearly becomes an illusion.

For programming languages, there are several steps that ensure correctness of the code. The first steps happen in the compiler. [Louden, 1997] presents a general overview of the compilation process, which is shown in Figure 5.1. The optimisation steps have been left out for the sake of

[1] The distinction is somewhat arbitrary. Is a shell script `/etc/init.d/` a part of configuration or is it a program? This classification depends on whether one has to write the script or manage the various scripts of the installation.

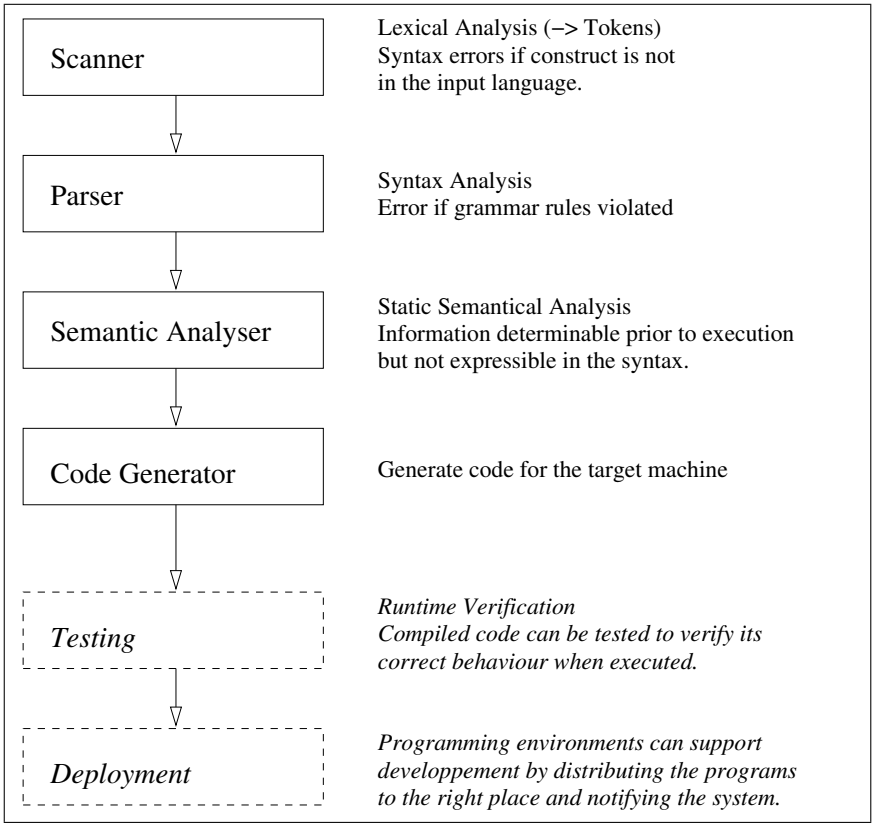


Figure 5.1: Steps of a compilation process [Louden, 1997]

simplicity, and the additional tasks of testing and deployment not mentioned by Louden have been added. The compiler checks the syntax and does a semantic analysis. The dynamic semantics of a program, however, cannot be determined by the compiler. To ensure that the program does what it is expected to and contains no security flaws, further measures have to be taken. In *Design by Contract*, special language constructs like pre- and post-conditions are used to logically prove correctness and to control correct program behaviour during runtime. Language constructs have to be built into the compiler to work. With *Runtime Verification*, expectations concerning program output and logfiles are provided. The expectations are constantly monitored while the real application is running. *Unit Testing* examines small, isolated parts of the code such as functions or objects. The concepts of unit testing and runtime verification apply to any programming language and require no special language constructs. The code is compiled and executed like normal programs, using the normal compiler. Last but not least is the deployment step, which is not a task performed by the compiler, but still part of the development toolchain. This might include generating an installer package that will prepare the system for the application, or upload the package to an application server and tell it about the new version.

A simplistic C method illustrates the different categories of errors that can be encountered in a program. Listing 3 shows the complete method, which basically adds the integer values provided by two pointer parameters. To avoid a security problem, the pointers have to be validated using some method `check` first.

Listing 3: The correct add method

```
int add(int* a, int* b) {
  if (check(a) && check(b))
    return *a + *b;
  else
    return INT_ERROR;
}
```

After spoiling the tension by showing the final result, let us look at where code verification will reveal errors, starting at the very beginning of the compile process. Listing 4 shows an error that will stop compilation when tokenising the program code into chunks in the scanner: the character "a is not in the alphabet of standard C.

Listing 4: Syntax (Lexer: Illegal character "a)

```
int add(int* a, int* b) {
  "a;
  return *a + *b;
}
```

When the line with the "a is removed, the scanner will find no more errors. In the next step, the parser tries to make sense of the tokens. In Listing 5, there is an error that will show in the parsing phase: a missing semicolon ';' after an instruction.

Listing 5: Syntax (Parser: Missing semicolon)

```
int add(int* a, int* b) {
  return *a + *b
}
```

Static semantics are violated in Listing 6. The return type of the addition of two integer values produces an integer, but the method is declared to return a floating point value. Semantic analysis will reveal this mistake.

Listing 6: Semantic (Wrong return type)

```
float add(int* a, int* b) {
  return *a + *b;
}
```

A C compiler has no notion of the meaning of method. There are further requirements which have to be tested by other means.

A human would expect a method called `add` to calculate the sum of the values pointed to by the two parameters, but this is not the case in Listing 7. It is important to ensure that the method actually does what it is expected to. While Listing 7 is trivial, there are many cases in real life that require unit tests or other methods to ensure correct operation. It is important that correctness testing happens automatically, once the developer has defined her requirements.

Listing 7: Requirement (Minus instead of plus)

```
int add(int* a, int* b) {
  return *a - *b;
}
```

The code in Listing 8 will produce the expected result whenever the parameters a and b are correct. However, it does not fulfil all requirements. There is no check for the validity of the parameters, which might cause random crashes or open a hole through which a hacker could attack the application. Some security aspects can be tested with according unit tests. Further measures like an analysis with a vulnerability scanner can be used to reveal additional problems [Viega et al., 2000].

Listing 8: Security (No pointer check)

```
int add(int* a, int* b) {
  return *a + *b;
}
```

After all of the aspects are satisfied, the final code will look like the one in the first listing. Note that the code generation step of Figure 5.1 has no counterpart in configuration. Configuration is not usually translated into another representation, but directly interpreted by the application. For the Verinec project however, there is a translation step from the XML configuration into implementation-specific formats. Configuration translation is discussed in Chapter 6.

After this excursion into code verification, we move back to configuration data. The compile-time steps, syntax and semantic analysis, are similar to steps that take place during the analysis of configuration. Configuration data is also tokenised, parsed and can be analysed statically. Runtime behaviour is not easy to test, possibly even more difficult than code behaviour. Remember, Verinec is not about testing applications. The question is not whether an application can correctly read its configuration data. What the administrator needs to know is whether a specific configuration will allow the network to operate correctly. Special tools are required for this last step in verifying configuration correctness.

Static rules can be used for the semantic analysis. With a knowledge of protocol definitions, some properties involving more than one device can also be verified with rules, making more extensive analysis possible than the semantic analysis in the compiler. However, as with program code, where many errors can only be detected during runtime, not all of the requirements for configuration can be coded with static rules. There are two methods to extend verification: testing runtime properties of the configuration, which is similar to unit tests and runtime verification in software engineering, and the formal verification of all properties of the configuration. The deployment, for which I will use the term adaptation here, must happen automatically in order to avoid introducing new errors. This is discussed in Chapter 6. The following sections discuss each step in configuration verification.

5.2 Syntax Checking

If a network device is provided with incorrectly formatted configuration data, it will very likely not behave as expected or refuse to work at all.[2] Depending on the format of the configuration, the quality of the parser and the place where the error occurs, either some parameters or the complete configuration file becomes unusable. If the device cannot revert to an older, correct configuration, service interruption is inevitable. Syntax errors are particularly likely when configuration is edited manually. But NMS which simply pass configuration data to the devices without syntax checking are subject to the problem as well.

The documentation of a network device should specify the correct syntax for its configuration. This information has to be translated into suitable rules. In a programming language, the lexer and parser detect syntax errors. An NMS does not necessarily have to parse configuration data, it can just transfer the data to the target device. It would, however, be reasonable to check the syntax. There are standard syntax specification languages to define file formats. Syntax checkers exist that use a specification to validate documents. For example, text formats can be specified using the Enhanced Backus-Naur Form (EBNF) [ISO, 1996]. When using XML, the standards DTD or XML Schema are appropriate.

If documentation is available, the syntax of configuration data is relatively easy to validate. A good system documentation should accurately describe the configuration format. For XML configuration, the best description is an XML Schema provided by the manufacturer, directly allowing validation with most XML processors.

The Verinec project does not use native configuration formats, but has its own XML language which is translated for the target devices (see Chapter 6). The internal configuration format is controlled with the help of an XML Schema. When the generation of configuration for the target devices is automated, the risk of syntax errors is eliminated – as long as this translation is implemented correctly.

[2] Which is worse depends on the situation. In a high availability setting, a device stopping on the slightest mistake in its settings is not desired. In a security sensitive environment however, it might be better if a firewall blocks all traffic instead of operating on an incomplete ruleset. Syntax errors can pass unnoticed if the network continues to operate.

5.3 Semantic Analysis

Semantic configuration rules are similar to the definitions for semantic analysis in compilers. They define the universal rules for the services employed. For example, the default gateway of a machine always has to be existing within a network for the configuration to make sense. Two network elements having a link between their interfaces need to use the same protocol to communicate. The semantic analysis is applied to the configuration data. It does not execute the configuration. Depending on the configuration format, a formalism to specify the rules has to be chosen.

Requirements that are not universally valid are not in the category of static semantics. Usually, compliance with policies is not expressed as a universal rule, although this can often be checked with a similar instrument as that used for semantic analysis.

5.4 Additional Rule Checking

Certain problems can arise which do not violate general semantic rules, but are specific to an organisation or to a particular network setup. Best practises or company-specific rules such as naming conventions fall into this category. The actual network indicates what further requirements must be met. For example, the configuration of a backup router could be different from the configuration of its productive sibling. In this case, should the productive router fail, the backup will not be able to take over transparently.

In a network that has grown over time and has been administered manually, there are very likely to be some inconsistencies. Testing whether a device is set up correctly for its task can reveal hidden problems. For example, a computer in the role of a server should normally not participate in a dynamic routing protocol [Dooley, 2002], passwords for user authentication should not be easily guessed and so on. The same concept of rules as those used for semantic analysis can be employed for tests of this kind.

Importing can also produce dangerous configuration if some features are not recognised by Verinec or are unsupported. Section 9.2.1 discusses parsing packet filter rules. One of the rules causes all packets to be accepted if they belonged to a protocol not supported by Verinec. This resulted in a rule that unconditionally accepted all packets. A test for rules that have an empty match list but are not the last rule in the chain would alert against situations like this.

In some situations, several devices or the whole network have to be taken into account, which makes matters more complicated. The use of a central configuration approach, as in the Verinec project, is well suited to this kind of analysis. Having accurate knowledge of the complete network makes it possible to cross-check whether the settings for different devices are consistent. If two routers use different routing protocols on a common link, this will lead to problems. Parts of this analysis can only be performed if the network model is enriched with meta-information. The application needs to know about the roles of the devices and about environments they are in. Implementing meta-information is discussed in Section 8.7.

5.5 Verification through Simulation

Defining rules that test correctness by specifying which combinations of settings are allowed and by defining referential integrity can become quite complex. Whether a service can be carried out successfully or not depends on many factors. To illustrate this, consider the example of a web server. In order to allow clients to access web pages, the server must be reachable from the Internet. However, to provide the requested information, the web server might need access to other servers, i.e. databases or file servers. The database might use yet other servers for authentication, and so on. If one uses only rules to test the configuration, this complexity has to be modelled manually for each service. Should even one sub-interaction be neglected, the configuration cannot be properly tested.

It is better to use behavioural tests on interactions of this kind, analogous to the use of unit tests in order to ensure correct program code. Network simulation can be used to verify network configuration [Jungo et al., 2004]. In the field of programming languages, the originators of the Extreme Programming concept [Crispin and House, 2002] promote a technique called **unit testing**.

[Jungo et al., 2005] propose the use of a network simulator to do unit tests on network configuration. A network test is composed of input events for the simulation and requirements that verify the output of the simulation. If the simulation creates the expected output, everything is fine. Otherwise, the configuration will not ensure the intended functionality of the network. With the network tests, it is possible to ensure policies such as "This machine may not access hosts on the Internet, but must have access to the intranet web server" or to run a simulated port scan attack on the whole network and see what information could be gathered. Tests can both require something to work or something to be blocked. Once defined, the tests can be automatically applied to a set of devices. Using automated tests, one can test the complete network with very little effort every time a change is made. This avoids unwanted side effects when changing configuration. The implementation of this concept in Verinec is discussed in Section 8.7.

Using simulation to verify the configuration is by no means a formal mathematical verification. This process is something between runtime verification and unit testing. [Barringer et al., 2004] defines runtime verification as using an observer to monitor the execution of a process and checking its conformity with a requirement specification. Runtime verification is used to verify the behaviour of the complete application in productive operation. When we test a network, it is not practicable to run the real network to check whether it is behaving correctly. While pure unit tests would examine isolated chunks of configuration, Verinec simulates the network as a whole. The correctness of the test results depends vitally on the quality of the network simulator used. However, if the network simulation is operating correctly, reasonably high trust in the correctness of the network can be achieved. While the greatest accuracy would be achieved by observing the productive network, this would require to first configure the productive network with the new, yet untested and possibly faulty configuration. Using a copy of the network for running tests is both too much trouble and too expensive.

What's more, using a simulator also avoids some of the problems inherent in real networks and permits one to do things that could never be done with a real network. One example is given in the next section. In log file analysis, it is very difficult to determine the relations between events in different network elements or even in different programs and layers of the same node. Each log file is generated in isolation from events that occur in other locations. A simulator that looks at the complete network can preserve the relations between events on the network. Relating the events correctly is crucial when analysing the source of a problem.

5.6 Testing Redundant Backup Strategies

In the previous section we discussed how simulating the network and checking the configuration with a unit testing concept can be used to identify misconfiguration and potential risks of intrusion. The network testing framework opens an interesting field of additional possibilities, some of which are impossible to do in a real network. Even if the configuration is perfect, network hardware is bound to fail sooner or later. To avoid service interruption, important parts of the network have to be duplicated, so that spare parts are available when something fails. There has been a lot of research on ensuring redundant paths in case of link failure, for example [Blouin et al., 2003]. In this thesis, the focus is not on link availability, but on whether the configuration of redundant devices results in correct behaviour after a device failure. This section is based on the paper [Buchmann et al., 2007a].

The behaviour of the productive network after a device failure could be tested at a time when disrupting the network does not hurt. An administrator would work during the night, unplugging devices and checking whether the network fails or not. In larger networks or in the context of the Internet, the network is always in use and disrupting it for tests is no option. Even if the administrators could do tests on the productive network late at the night or during weekends, this method is cumbersome, to say the least. Moreover, when a network is planned, certain hardware might not yet be available. A slightly better option is to build a separate network for testing purposes. Failures can be produced here without any risk for the productive network. However, it is expensive to have extra hardware just for testing. Another problem is that the configurations of the testing and production networks can differ. Even small differences might result in a mistake being overlooked that could interupt the network in case of a failure. In practice, it is impossible to test all of the possible failure constellations of a large network in this way, it would simply take

too long. Simulating the network allows to experiment with all possible combinations of failures. In the simulation, any device or even combinations of two or more devices can be switched off to check whether the network would still function as it should. By automating this process, it is possible to identify single points of failure and evaluate whether backup solutions and redundant hardware are used correctly. This would satisfy the use case on redundant failover from Section 4.7. [Vasseur et al., 2004] distinguishes between planned and unplanned network failures. Planned downtimes happen for purposes of maintenance and provisions can be made beforehand. This discussion focuses on unplanned downtimes because of hardware defects or bugs.

Research has been done on calculating expected system failure probabilities, either using an analytic approach (for example [Chang et al., 2004, Chen and Yuang, 1995, Yeh et al., 2002]) or by simulation [Cancela and Khadiri, 1996]. Analytic solutions use mathematical methods to give exact answers. [Law and McComas, 1996] have written an introduction to the process of deciding in which situations simulation techniques should be preferred over analytic solutions. They state that the analytic approach is not adequate for complex network topologies and protocols, but can only give mean values of expected performance and requires considerable mathematical sophistication from an analyst. Failure probabilities help to give an idea of the reliability of a network. Calculating failure frequencies does not, in itself, improve the reliability of a network. It is necessary to identify to what extent the network depends on which nodes and how the situation can be ameliorated. Furthermore, simple connectivity between server and client nodes does not mean that the server nodes can provide the expected services. Some operations might require the server to access some third node, for example a DNS or file server or an Internet gateway.

To test fault tolerance, the network requirement tests can be re-used. Instead of doing just one simulation run, the simulation is repeated with each node switched off. In more complex cases other nodes could stop operating while the original faulty one is still broken.[3] Therefore all combinations of switched off nodes should be tested. In this way, one can determine the node or combinations of nodes that cause failures for each test.[4] By repeating the network test cases for every failure combination, one can see which nodes are important for the operation of the network and single points of failure can be identified. If the redundancy is high enough, single node errors do not cause any tests to fail. However, when enough nodes fail, any network is interrupted. One needs to find a good compromise between reliability, complexity and costs. Simulating the network and testing fault tolerance helps to determine crucial nodes in the infrastructure. It allows one to quickly test what effects additional redundancy would have. With this information, the network can be improved in the most fragile areas first.

A network element is actually composed of several interfaces and services which can crash individually; in addition to this, the network connections can also break. With some redundancy, the failure of only one link or interface sometimes has less impact on the network than the malfunctioning of a complete device. The effects of a broken interface or link also become visible during a simulation if the device on the other end of the link is switched off. Thus, the failure of complete nodes is a kind of a worst-case scenario. The added level of detail, however, might help to identify inexpensive adjustments that can considerably improve the overall failsafeness of the network.

Devices can stop providing their expected services [Dooley, 2002] for different reasons. He speaks of *hard failures* when the hardware breaks or software crashes. A device can also stop operating because of congestion or traffic anomalies, which is called a *soft failure*. As Verinec does not consider network capacity, this section will be limited to hard failures.

5.6.1 Unequal importance of devices

Defining a lot of tests and simulating many malfunctioning devices leads to a lot of test failures. Because of limited time and budgets for equipment, an administrator should concentrate on the most serious risks. This raises the question of how to assess the importance of failed tests.

Counting the number of users affected by a failure and the number of tests failing can give an

[3] Verinec does not consider external factors like power outages taking down a large number of nodes in this paper. Each node might fail at any time, but the failures are considered as being independent of each other.

[4] There is no need to switch nodes on and off during one simulation run to see if the routing protocols where properly designed. Verinec is not intended for protocol verification. Rather, it checks whether the network is operating properly, assuming the protocols to be well designed. It would, however, be possible to use the Verinec simulator for routing protocol verification.

idea of the consequences of a malfunctioning device. If only the Email system no longer works, it is not as bad as a complete network outage, meaning no Email plus no Web, no file servers and no printing. If a hundred users are affected by a problem, this is worse than only ten users being affected.

However, there are more complicated cases. Let us assume that a company has a local network, a DMZ with intranet and Internet servers and Internet connection through the DMZ. If the main Internet gateway fails, all of the users and servers are cut off the Internet. However, the users can still access the intranet. If the connection from users to the DMZ fails, the users can no longer work with the intranet, but the Internet servers are still reachable from outside. Here, the relative importance of the cases depends on the necessity of the intranet for local users versus the role of the Internet servers for customers and partners. Network access might not be essential to all users. Some might work mostly on a local machine or not often work at a computer at all. To take this into account, tests could be assigned an importance factor. A test for a client machine of a user who depends a lot on Internet access is weighted higher than a test for a computer that is currently unused. It also becomes possible to define how important Internet servers are, compared to workstations and other local equipment.

Running the tests shows which network elements' failures make the important tests fail. In this way, an administrator can judge how relevant each network section is and can concentrate on improving the most crucial sections. Later on we will see that the importance of failures can be combined with the probability of the same failures.

5.6.2 Complexity

In the previous sections, all possible combinations of malfunctioning devices have been considered. Each device is considered independent of the others. This brute force method has a complexity of $O(2^n)$, where n is the number of nodes in the network.[5] Each test means simulating a set of test cases in the network. Since this is not very scalable, the complexity has to be reduced.

In the introduction to this section we raised the question of whether one should test the failure of individual services or only complete node crashes. From the point of view of complexity, it doesn't matter. The number of nodes n is multiplied with the number of s services when all combinations of service failures are considered. $O(2^{n \cdot s})$ with `s` a constant factor is in the same complexity class as $O(2^n)$.

If one takes the actual meaning of the tests into consideration, one can limit the number of cases that must be simulated. If something does not work without node A, it is not necessary to run this test again with node A plus additional nodes malfunctioning, as it will not stop failing with less nodes functioning. This cuts down the number of simulation runs that are needed if something fails. The worst case complexity (which would in the present case mean a perfectly failsafe network) remains unchanged, meaning that no test fails even with almost all nodes malfunctioning.

A further measure would be to fix an upper limit for the number of nodes assumed to be malfunctioning at the same time. If more than a certain percentage, say 50%, of the nodes are out, it is very unlikely that the network is still operable, thus testing with even more nodes out is not necessary. This would reduce the number of cases to be tested by half. However, multiplying the number of cases with a constant factor does not change the complexity of the algorithm.

In most cases, a large number of nodes failing at the same time is no independent event. The probability for simultaneous failure of more than a few devices is extremely small. The risk of virus infections or external factors like an outage in a building's power supply system or a natural disaster is much higher. Emergency power supplies and survivability in the case of disaster must be carefully planned; however, factors that are external to networks do not lie within the scope of this thesis.

Avoiding unnecessary tests

Not all of the network elements perform tasks that are important for the network. Most of the machines in a typical campus or company network are client machines. By definition, clients do not provide network services and their failures only affect the user of the machine, but no other part of

[5] Choosing all combinations to fail from n nodes: $\sum_{k=0}^{n} \binom{n}{k} = 2^n$.

the network. Thus all simulation runs that include malfunctioning network elements in client roles can be skipped. In order to do this, additional information about the roles of network elements is required. This information can be provided by a role model that adds a type description to each network element, as proposed in the use case in Section 4.5.

A generalisation of this concept can even do without a role model. Only the nodes that actually have something to do need to be simulated as malfunctioning. If a node neither processes nor sends any packets during a simulation run, the failure of that node will not be noticed. There is no need to simulate the same constellation with an unused node switched off. The proposed algorithm works as follows:

- The simulations are arranged by number of simultaneous failures, level 0 meaning no failures.
- Along with each simulation run, the set of nodes that are to be considered on the next level is established. A node that does something on a particular level might cause a to test fail if it fails itself.
- The iterative process starts with 0 nodes malfunctioning, meaning a network with no problems.
- All nodes that process an event will be simulated to fail as single nodes.
- For two failures at the same time, only nodes that had something to do when one node failed are taken into consideration, in combination with the failure of that node.
- This procedure is continued until there are no more successful tests.

Hence, the list of nodes taken into account grows as alternatives are used after node failures. Assuming a linear increase of "important" nodes in a growing network, this still does not change the basic complexity. However, in practice it reduces the number of tests considerably, delaying the point where the network becomes too large to be validated using the simulation method.

Mean time between failure

A reduction influencing the complexity can be found when considering the Mean Time Between Failure (MTBF). The MTBF is a statistical value, indicating the operating time after which half of the devices of a particular type can be expected to fail. Values for typical MTBF of nodes and links can be found in the relevant literature, for example in [Vasseur et al., 2004]. According to one observation, link failure depends on the length of cables. In this section we only look at node failures. Link failure is not inherently different, but would require taking cable length into account, which is not usually available in Verinec. Link failures are mostly important within the context of large backbones or provider networks, while Verinec is mostly aimed at local networks.

When calculating the probability of multiple simultaneous failures, the number of cases to be tested can be reduced dramatically. [Dooley, 2002] combines the probabilities of failure according to the MTBF of the devices with probabilities for the event of a number of devices failing simultaneously. He uses several simplifications, all of them overestimating the probability of failure. The most important of these is to assume that the failure rate is linear, reaching 50% of broken devices after the MTBF. This results in the probability of $\frac{1}{2 \cdot M}$ for a specific device to fail on a particular day, where M is the MTBF in days. In reality, the failure rate will be much lower until near the MTBF, where it increases rapidly.

According to Dooley, the probability for several devices to fail on the same day is

$$ {}_kp_n = \frac{n! \cdot (2m-1)^{n-k}}{k! \cdot (n-k)! \cdot (2m)^n} = \binom{n}{k} \cdot \frac{(2m-1)^{n-k}}{(2m)^n} $$

${}_kp_n$ is the probability of k failures on the same day in a network of n devices, with m being the MTBF measured in days.

An example can illustrate this. With n=1'000 nodes relevant for the network and a MTBF of m=2'000 days (approximately 5 years), the probability of k=10 nodes to fail on the same day is in the order of 10^{-13}. Even in a big network, the probability of more than a few nodes failing simultaneously is extremly small. Figure 5.2 shows the probability of k failures on the same day for a number of nodes n from 500 to 5'000. In a small network, no failure on a specific day is most probable. With an increasing number of nodes, failures are to be expected.

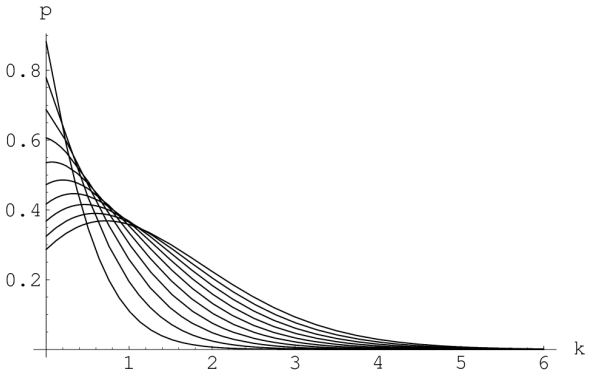


Figure 5.2: Plot for probability of k simultaneous failures out of n nodes, with 500 to 5'000 nodes.

It is legitimate to stop testing for cases with small ${}_kp_n$. At the point where the probability for external problems affecting the usage of the network becomes much higher than the probability of so many devices failing independently, further improvements on the network stop being reasonable. Table 5.1 indicates how k evolves to satisfy ${}_kp_n < 0.0001$, assuming a MTBF of m=2'000 days. The limit of 0.0001 means that the untested failure combinations can be expected to occur less than once in 10'000 days or less than once every 27 years. The third row indicates the order of magnitude for the number of combinations of k elements out of n $\binom{n}{k}$.

While limiting k reduces the number of combinations tremendously ($2^{1000} \approx 10^{301}$ while $\binom{1000}{5} \approx 10^{12}$), there are still many simulation runs required for large networks with 1'000 nodes or more. Therefore, it is useful to combine this limit with the previously mentioned mechanisms to avoid unnecessary test runs.

n	100	500	1000	2000	3000	4000	5000
k	3	4	5	6	7	8	8
$\binom{n}{k}$	10^5	10^9	10^{12}	10^{16}	10^{20}	10^{24}	10^{25}

Table 5.1: Values of k for some n to have ${}_kp_n < 0.0001$

An improved selection of device combinations

In the previous discussion, we assumed that the Mean Time Between Failure (MTBF) is the same for all devices. If it varies considerably between the devices, it is possible to improve the selection of combinations needing to be tested. For this, the MTBF must be known individually for each device model.

Still assuming a linear distribution of the failures, the probability for a specific device to fail on a certain day is $\frac{1}{2 \cdot M_d}$, with M_d being the MTBF for that device in days. Combining these probabilities gives the probability for any combination of failures as

$$\prod_d \frac{1}{2 \cdot M_d}$$

where d takes the values of each device in the combination.

Instead of setting a limit to the number of devices failing simultaneously, the minimum probability for a combination to fail is now limited. Combinations with a very low probability need not be examined.

When considering the probability of an event, it can also help to weight the importance of the tests as mentioned in Section 5.6.1 and combine it with the probability of failure. Some tests can be run for more combinations of device failure than others. If a test is very important, it is run even if the combined failure probability is low.

Besides giving a more precise selection of what must be examined, this method would also help to weight error reports. Dooley's method allows a rough estimation of how probable a situation is by counting the number of device failures at which a test fails. This makes an exact probability indication available for every situation. Weighted tests also show how problematic a failure would be. A single point of failure of an extremely reliable device might be less dangerous than two redundant but very unreliable devices.

There is, of course, a major drawback to this method: the administrator has to specify the MTBF for every device used in the network. In a homogeneous network with three or four types of devices, this would be feasible. However, in a heterogeneous environment, the work involved can quickly exceed the benefits gained by the more detailed simulation.

5.6.3 Network Reliability Case Study

To illustrate how redundancy testing can improve the network configuration, let us look again at the use case from Section 4.7. To test the network from Figure 5.3, the administrator has to define four test cases for a node in the Internet, requiring that it must be able to reach each of the servers. The simulation yields the same result as the use case example: when either switch or router is out, the node on the Internet can no longer reach any server and all tests fail.

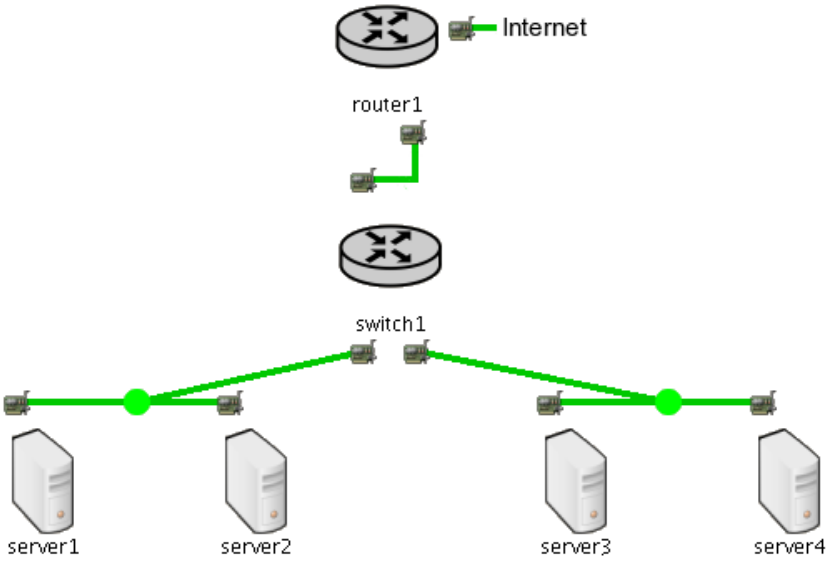


Figure 5.3: Network without redundancy

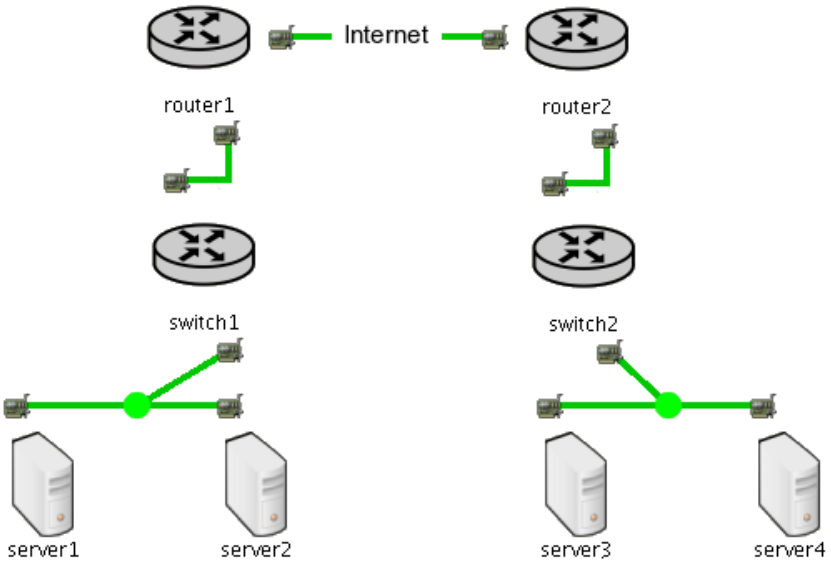


Figure 5.4: A bad attempt to improve redundancy

If each server acts independently, the services of a node are no longer available when it crashes. Hence, the servers should be configured redundantly, meaning that they can take over the tasks of a malfunctioning machine. This example focuses on the reliability of the network connection and assumes the server nodes to be configured as correctly as possible.

One strategy for the elimination of single points of failure is duplication. Figure 5.4 illustrates

the new network after duplicating routers and switches. There are two switches, each of them connected to a separate router. There remain no single points of failure.

In this case, the simulation will show that reliability is not much improved. In fact, the probability that all four servers are reachable has even decreased! If one of the switches or routers fails, half of the servers become disconnected and some tests, but not all, fail. There are more switches and routers in total, thus the probability that one of the devices fails is increased. However, it is less probable that all four servers would become disconnected: that only happens when both routers fail; in the original design, one failure disconnected all four servers.

The servers also run a higher risk of loosing the connection to each other now, since they have lost the direct connection through one common switch in the original design. Assuming that they need to interact to perform their service[6], the simulation detects that a breakdown of a part of the network will actually result in a complete failure for some services. Even if each server is completely independent of the others, a small change will result in a vastly improved reliability of the whole network.

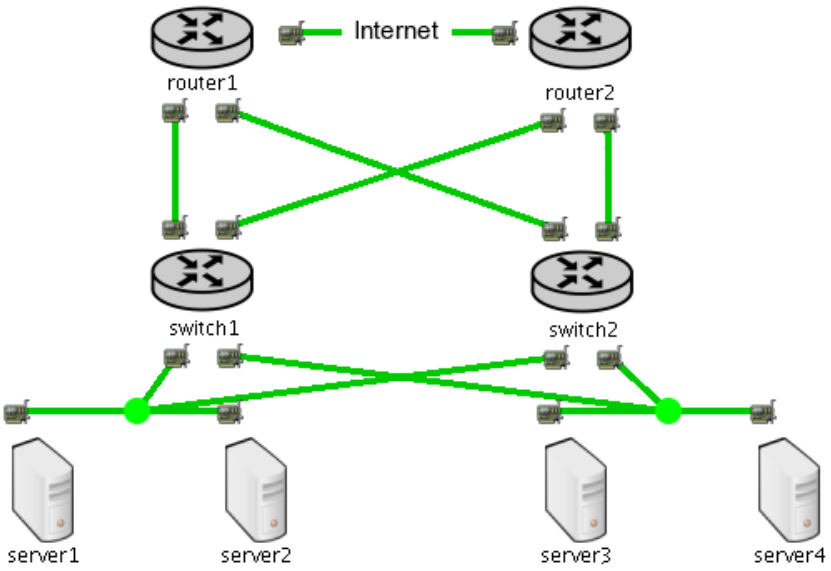


Figure 5.5: Fully redundant network

The final state is illustrated in Figure 5.5. Each switch is connected to both server pools and to both routers. If one switch fails, the other can take over all operations. Now the reliability has significantly increased. Any single device failure will not affect the network. Even the malfunctioning of one router and one switch at the same time can be handled without any of the tests failing.

The reliability of the communication between the servers would further be improved by a link between `switch1` and `switch2` for cases where switch1 looses one part of the servers and switch2 the other part. With an inter-switch link, the network would not be cut apart by a single link failure. If the simulation takes interface and link failures into consideration, this improvement would also be rewarded with a better reliability outcome of the simulation.

5.6.4 Conclusions on redundancy simulation

While a straightforward approach to examine multiple simultaneous failures is the simulation of every possible combination of malfunctioning devices, practical reason suggests one should limit the number of simultaneous failures. At the same time, taking the probabilities into consideration helps to weight the importance of failure combinations. Identifying the problems posed by several

[6] E.g. if the web server runs on a different machine than the database, and has a separate file server to store the pages.

devices failing simultaneously produces a lot of output, which is simply confusing if it is not accompanied with probabilities or other information on the relevance of each problem.

There is room for further research. This section only discussed the basic question of whether a network can still operate after device failures. Traffic load or load balancing issues have not been considered. Often the redundant devices are all in use during normal operation. A failure of one device increases the load on the others. If the network is operating at its limit already, it can break down because of congestion even if there is still some connectivity. Even worse, local failures can affect the whole network as traffic is redirected. These issues are discussed, for example, in [Milbrandt et al., 2007]. It would be interesting to consider traffic load in the simulation. In that case, tests could not only check whether the network could theoretically continue to operate, but also whether the congestion management is set up properly and all relevant traffic can still be handled.

5.7 Related Work in Configuration Verification

This section discusses different approaches to ensuring correct network configuration. A discussion of the various approaches to network management in general is found in Chapter 2.

The idea of abstracting from implementation-specific formats is used in the Netopeer project [Lhotka and Novotny, 2007] as well as in Verinec. [Matuska, 2004] formulates the idea of using simulation to verify the correctness of Netopeer configuration. Automated translation from abstract configuration into implementation-specific formats and automatic distribution is crucial for any system verifying configuration. Manually configuring devices after a verification process renders the verification useless. Typing errors or copy-paste mistakes are likely to happen. Handling mistakes could also happen. A correct configuration on the wrong machine will not produce the desired result. And finally, it would be very dull work for a human to copy configuration to devices manually.

[Bhargavan et al., 2002] analyse the output of a network simulator. However, they are focussing on protocol verification, whereas Verinec assumes the protocols to work properly and is concerned with network planning mistakes.

Sanjai Narain proposes an alternative to simulation for configuration verification: Service Grammar [Narain et al., 2003]. Service Grammar is an abstract representation of configuration data as logical information. Narain's approach is to use logical reasoning on the configuration in order to determine its validity. Similar to what is done in Verinec, he abstracts from implementation-specific formats to examine the features of the underlying standard. Narain suggests specifying the end-to-end functionality and using formal verification to ensure that the underlying services are set up correctly. He outlines an approach to developing formalised tools for global reasoning on the logical structure of the whole network system. To avoid having to prove the complete protocol stack in one step, he looks at each service separately and works on the premise that the network is configured correctly if this is true for each service individually.

The Verinec approach does not rely on the formal verification of individual services. Instead, simulation is used to test whether the requirements for the network are fulfilled. Tests for the simulation are easier to define than formal verification, as they need a less thorough understanding of the underlying protocols. Simulation also reveals the impact that problems with one service can have on other services depending on it. As discussed in Section 4.6, a server might need to use other services in order to provide its services. Service grammar assumes that the network is correctly configured if every service is correctly set up. For the administrator to be sure that a service will be able to operate, either the complete network configuration must be correct, or all dependencies must be modelled by hand. The later is not suggested by Narain. Modelling by hand is error-prone and thus not suited to verify the correctnes of configuration. Having a completely correct network would be nice – but in real life, network operators usually do not have enough time to correct every little detail of configuration. Simulation directly identifies the configuration errors that cause important failures.

The sub-problem of redundant backup paths discussed in Section 5.6 is sometimes addressed with mathematical methods. When the delay caused by activating a backup route becomes vital, these methods are important. Verinec assumes that dynamic routing will find a new route after a failure. However, the process of negotiating new routes takes time. *Completely Disjoint Backup*

Paths is a concept that requires alternative paths to be ready for use in case of failure. Since the paths do not share any links or nodes, the backup will still be usable if any part of the primary path breaks. There exist two levels: *Path Diversity* means that the paths are disjoint, but that the backup path may not provide as much capacity as the primary path. *Path Protection* is achieved when the disjoint backup path can accommodate the full capacity. [Herzberg and Raz, 2007] examine the survivability of a network. A straightforward method is to duplicate all of the paths with additional hardware. While this is fast and reliable, it is extremely expensive. For the concept of shared backup paths, several independent paths can rely on the same hardware for the backup path. However, shared paths cannot be reserved in advance, as the link cannot provide the necessary capacity for all backup paths at the same time. Herzberg et al. present a concept to pre-connect the shared backup paths, in order to speed up the path setup process when the path is needed. They also provide an algorithm to calculate shared backup paths ensuring path protection. The calculations take into account that co-operating network operators require fairness between capacity offered to others and capacity used by them.

The problem of link overload resulting from failures is also tackled by [Milbrandt et al., 2007]. Based on the expected traffic, they decide where over-provisioning of capacity, links or nodes is required for the network to survive failures. Alternately, service level agreements can be revised when over-provisioning is not possible. Risk discovery plays an important role in this estimation. Some failure types are usually correlated. According to Milbrandt et al., the most common reason for link failure is physical damage to the cables by excavation work. To adequately calculate the risks, inter-dependencies of failures have to be taken into account. For example, physical paths of cables or nodes in the same building have to be known.

If the specific problems of node and link failure can be disregarded, there are mature tools that estimate traffic loads. Two major simulation frameworks are the Scalable Simulation Framework (ssf) [Cowie et al., 1999] and the Network Simulator 2 (ns2) [ns, 2005]. With Verinec using XML as the network definition language, it is possible to export the network configuration into a format suitable for an existing simulation framework. There is a master student at the University of Fribourg working at translating Verinec configuration into ns2. This will make it possible to predict whether links will be capable of handling the (estimated) traffic.

As Verinec is aimed at network configuration management, the networks concerned will often be of medium size at most. A network with more than a few thousand nodes will probably not be administrated from a central location. This somewhat reduces the importance of the scalability issue inherent in the centralised approach. In the context of Grid computing and peer-to-peer networks, there are projects concerned with simulating huge networks consisting of several 10'000 nodes, for example [Liu and Chien, 2004] or ns2 [ns, 2005].

Chapter 6

Adapting Configuration to Network Devices

In analogy to what happens in a programming environment (see Section 5.1), configuration data has to be applied to the devices. Progams need to be notified about the configuration change, for example by restarting them. Verinec uses abstract configuration internally. Before configuration is applied to the device, it has to be translated into the implementation-specific format. Because all device configuration is translated, an administrator can define the configuration once and then generate it for all of the supported implementations. Theoretically, this allows one to exchange one machine for another, without needing to change any configuration data.

Automated translation and distribution of the configuration is absolutely necessary within the context of Verinec. The verification process would lose most of its advantages if the configuration had to be adapted to the devices by hand. In manual configuration, typing errors or other mistakes could render the carefully verified configuration faulty again.

In this thesis, the process of translating and applying abstract configuration to a network is called *adaptation*. The process can be divided into three steps. Some implementations of a service might not support everything that can be expressed with a chosen abstract XML configuration. The first step, called *restriction*, produces warnings about features of a given configuration that cannot be meaningfully translated for the selected architecture. After problems of this nature have been solved, the second step is the *translation*. XML configuration is transformed into instructions or configuration files specific to the target system, like the contents of a file or SNMP instructions to execute. The last step is distributing the configuration to the target systems by using a suitable mechanism. Figure 6.1 illustrates the adaptation process.

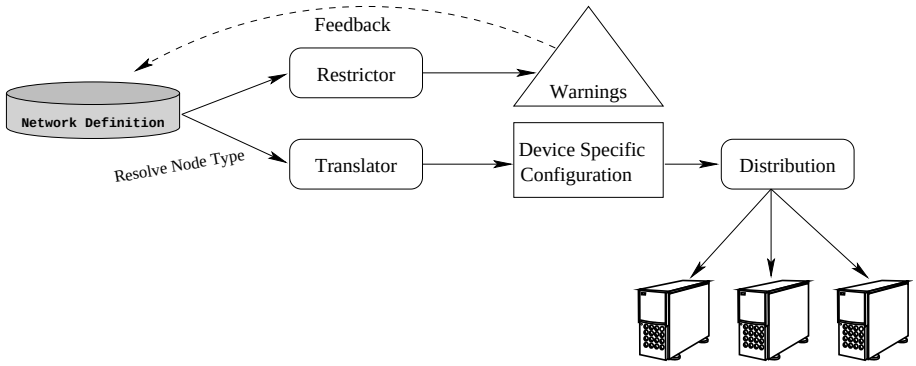


Figure 6.1: The adaptation process.

Additional information is necessary for the adaptation process. Each machine provides a couple of services. There might be many implementations of the same service found in one network. For example, packet-filtering under Linux can be done using either iptables, ipchains or ipfilter. Cisco routers have their own packet filtering service and so on. The NMS needs indications that

show which implementations of each service is used to translate the configuration accordingly. It needs further indications showing how to access the devices in order to configure them. This thesis proposes to use a system of types to specify the implementations of services. Collections of translator specifications for different services can be grouped into a *node type*.

Similar to the definition of types, we need to create a structure that will define how the configuration data reaches the system to be configured. *Target* information defines how to access systems that must be configured. They can specify a path to copy files to the right directory (i.e. a directory mounted from the server) or server name and the remote directory to copy configuration files over the network. Other options include SNMP and similar remote management protocols.

6.1 Restriction and Translation Steps

When the NMS has determined the device type, it knows which translator and restrictor to use for each service. The restrictor identifies constructs in the configuration data that are not usable with the selected implementation. It can produce human readable explanations for each construct that the translator will not be able to handle. This helps an administrator to adapt the configuration to remove unsupported constructs. A restrictor could also transform the configuration into a new configuration without the problematical constructs. The second variant would be useful in the verification process. A simulation of the full configuration cannot detect problems that arise from incomplete translations. It would be best to simulate exactly the configuration that will be on the real device.

After the restriction information has been determined and the user warned about potential problems - or else, after all problems have been solved, the translation can be performed. The translator takes the abstract configuration of a service and produces an implementation-specific format. Unsupported constructs are ignored, resulting in information losses if such constructs have not been eliminated by the user during the restriction phase. Possible output formats for the translations can be another XML format, plain text files in the proprietary format of a service implementation, or a series of commands for SNMP, WBEM or some other remote management standard. An appropriate distributor that understands the output format has to be chosen.

The adaptation framework for Verinec is discussed in Section 8.4. Implementations of translators and restrictors for various services are presented in Section 8.5.

6.2 Distributing the Configuration

The resulting, device-specific configuration is distributed onto the systems with the use of targets. There are two philosophies for updates: push and pull. With pull, devices regularly ask for new information. In push settings, devices are informed by the management station about the new configuration. Pulling is most useful for tasks like software updates, where there is a large number of devices and the updates are not extremely time-critical. For configuring devices, push is more appropriate. Configuration changes become visible immediately and there is no overhead traffic when nothing changes.

There are different proprietary protocols in use for remote configuration. There are also some common standards that can be useful, for example SNMP and WBEM. Many protocols require an agent to run on the target system. In the case of Verinec, one of the design goals was to avoid having to install agents on the target systems. A plugin architecture has been designed that uses various means of getting the configuration to the system. The plugin to use is specified by the *target* information of the node type.

Some distribution methods can just overwrite configuration files with the new configurations. Others need to modify settings and store the new configuration. If the target system does not monitor the configuration data for modification, the distribution process has to take care to notify it of the new configuration. Often, individual services or the whole system have to be restarted to load the new configuration. Some services also need to be stopped before the configuration can be updated. To do this, a system of pre- and post-commandos could be used. These commands are executed by the distributor before respectively after the configuration update takes place. Commands are specific to the device implementation and the target. Section 8.4.3 presents the implementation of the distribution framework in Verinec.

6.2.1 File Copies

In the Unix and Linux world, many services use configuration files placed in specific locations in the file system. Such service implementations are straightforward and easy to support. To create a configuration file, the following information is required:

- Content of the file
- Path and filename

The file contents are generated by the XSLT translator in the implementation-specific configuration format. The path can be determined in two places: The translator can provide a default path for the configuration file of a service implementation. The target should be able to overwrite this path. Various Linux distributions have been very creative when naming the configuration files under `/etc`). Although the file contents are identical, the names are different. It would be inelegant to duplicate the translator simply to change the filename.

The target should also permit the specification of a path prefix for a target system. This would enable the administrator to mount the target machine's file system onto the file system of the local machine and just copy the generated files to the appropriate locations, relative to the mount point.

The application of file copies can involve copying onto a local file system, as well as to remote machines. The file transfer function of the popular ssh is a good way of doing remote file copies. Most Unix and Linux systems allow connections by ssh, making the setup straightforward.

Pre- and post-commands are simple: the commands can be executed as shell commands, either locally, or using the ssh connection. The translator knows the target system and standard locations of files, allowing it to specify the correct parameters. However, care should be taken not to call unconventional programs. All programs to be used should either be part of the base installation of the system or part of the service implementation to be configured. The implementation of a distributor for local file copies is presented in Section 8.6.2, a distributor for remote file copies in Section 8.6.3.

6.2.2 Simple Network Management Protocol

SNMP is a standard allowing to monitor and manage devices in a network. An introduction to SNMP is provided by [Ehret, 2005], an excellent reference in [Mauro and Schmidt, 2001]. It consists of a protocol and a definition of databases for network configuration called Management Information Base (MIB). A typical exchange in SNMP is depicted in Figure 6.2. The manager contacts an agent and requests some information. The agent communicates with the device and sends the result back to the manager. Thus the manager acts as a client, while each managed device acts as a server for SNMP requests. Requests can either ask for information or set values, or both. The agents carry out the requests, interacting with the device.

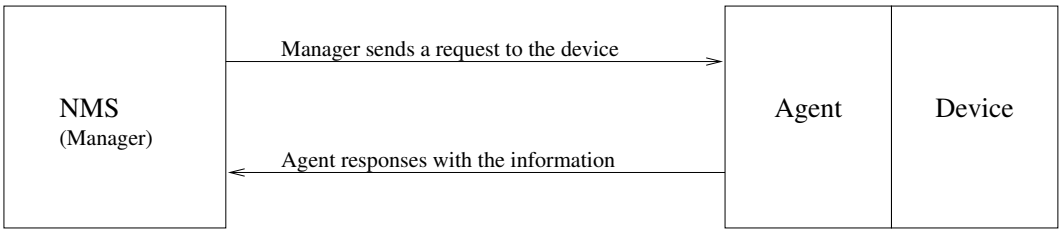


Figure 6.2: SNMP concept.

MIBs define the structure and semantics of the information. Information is identified using a tree structure. The root `iso.org.dod.internet`, or in numerical representation 1.3.6.1. From there, `mgmt` starts a tree of standard MIBs, common to all SNMP agents, while the `private` branch is used to group network companies needing to define their proprietary MIBs. This scheme results in rather long numbers for identifying specific values in a MIB. The agent acts as a translator between the internal configuration of the device and the SNMP standard. A vast selection of MIBs

exists for all kinds of information. Vendors are also allowed to create their own MIBs, which can support special features of a device.

Several versions of SNMP are in use. SNMPv1 was the initial version. It uses unencrypted transmission, sending even the password in clear text over the network. The password is called *community string*. Three different passwords can be set for "read only", "read and write" and agent initiated communication ("traps"). The second version, called SNMPv2, contains a very complex security concept that made it hard to adapt for device manufacturers. A simplified version that relies on the community strings as used in SNMPv1 has been introduced as SNMPv2c. This version is still supported by a lot of devices. SNMPv3 finally solved the security issues by providing a concept that is less complex, but reliable. It supports MD5 and SHA for password encryption and DES and AES to encrypt the SNMP commands.

SNMP development put a lot of effort into defining proper encoding and formats. This was necessary to allow SNMP to reliably operate with all kinds of network technologies and operating systems. Using an existing transport protocool to define encodings, for example HTTP and SOAP, would have made the standard much more compact. But network devices usually do not have the required processing power and storage capacity to process complex XML documents.

In Verinec, the translator could create batch files or generate XML output to script a SNMP session. However, this was not implemented, as most devices cannot be completely configured using only SNMP. In his master thesis, [Ehret, 2005] showed that SNMP is mostly used for network monitoring. Modifying configuration is not well supported by the implementations. Information is usually read only and has to be modified by some implementation-specific method.

6.2.3 Cisco

As an example of network appliances, Cisco[1] routers are discussed in more detail here. Cisco was founded back in 1984 by two computer scientists at Stanford University. From the beginning, it produced gateways and routers to interconnect networks. It has since become one of the big players in network equipment manufacturing, with a revenue of $24.8 billion and 38,413 employees [Cisco Systems, 2005].

Cisco devices run the Internet Operating System (IOS) developed specifically by Cisco. Configuration is both represented as and manipulated by lists of commands. The device distinguishes two configurations. One is the saved configuration that is loaded after rebooting the device, the other reflects the current state. *startup-config* is read when the device is started. It represents the initial configuration of the device after a reset. *running-config* reflects the current configuration of the device. Configuration is changed by applying rules either to the running state or to the startup configuration. The exact syntax and the available options depend both on the type of device and its capabilities, as well as on the IOS version used. For more detail, the reader is referred to the according documentation. An introduction is to be found in [Ehret, 2005], [Cisco Systems, 2003] provides the technical documentation. The remainder of this section discusses how Cisco devices can be remotely configured.

The standard way of changing configuration on Cisco routers is using a command line interface over telnet or ssh. Some devices also contain a web server that allows basic configuration via a browser. Cisco devices are also capable of up- and downloading their configuration to and from a server. Neither interactive shells nor a web interface are practical options for an automated configuration tool. The SNMP support of the Cisco router model 831 examined by [Ehret, 2005] is basically limited to read-only. However, it is possible to use SNMP to trigger the Cisco device to download or upload new configuration files through Trivial FTP (tftp). This file transfer is the most promising way of automatically configuring a Cisco device. The SNMP agent supports the MIB CISCO-CONFIG-COPY.[2] Unfortunately, the MIB does not permit a direct transfer of the contents of configuration files over SNMP. It can only be used to trigger the device to download the configuration file from a server. There are different protocols to choose from, but the Cisco device implements only the client side of each of them, needing a server to run somewhere with the configuration files.

[1] `www.cisco.com/`.

[2] CISCO-CONFIG-COPY is available since IOS version 12.3-11.T3. Before, other MIBs like CISCO-FLASH or OLD-CISCO-SYSTEM were available. They are less powerful and if the CONFIG-COPY MIB is available, the others are deprecated. The research was therefore restricted to CISCO-CONFIG-COPY.

The configuration MIB defines one important table, the ccCopyTable. It contains a row for each configuration transfer that is underway. As is usual with SNMP, the rows are used both to specify parameters and to retrieve information about the transfer state. Table 6.1 gives an overview of the important input parameters that trigger copying a configuration file. All parameter names begin with `ccCopy`, which has been omitted in the table for the sake of readability. The numbers of the possible values are what is actually passed on over SNMP, the names are used to explain the meaning of the parameters.

SNMP Parameter	Possible Values	Description
Index	*unsigned int*	Index in the table of copies currently being performed
Protocol	tftp(1), ftp(2), rcp(3), scp(4), sftp(5)	Specify which protocol to employ
SourceFileType DestFileType	networkFile(1), iosFile(2), startupConfig(3), runningConfig(4), terminal(5)	Which file to operate on
ServerAddress	*IP address*	IP address of server to download or upload configuration file
FileName	*String*	File name of the configuration to use
UserName UserPassword	*String*	Credentials for logging into the server (tftp and rcp protocols do not require a password)
NotificationOn\ Completion	*boolean*	Whether a notification trap should be sent after transfer has been completed
EntryRow\ Status	active(1), notService(2), notReady(3), createAndGo(4), createAndWait(5), destroy(6)	Used to start transfer or remove table rows

Table 6.1: CISCO-CONFIG-COPY parameters.

Using SourceFileType and DestFileType, it is possible to copy the running-config to the startup-config or replace a bad running-config with the saved state. For remote configuration, SourceFileType has to be networkFile, for backups, the DestFileType has to be networkFile. At least, either the source or the target have to be of the type runningConfig or startupConfig. The new configuration file does not necessarily overwrite the existing configuration, but is merely a script executed within the current state of the running or startup configurations respectively. To replace access-filter rules, for example, they first have to be deleted with a `no access-list` statement.

The transfer is prepared by creating a new row in the copy table. When all parameters are set, the transfer is started by setting the EntryRowStatus to active. Afterwards, the status cannot be modified until the transfer is completed (either successfully or failed). While experimenting with the Cisco 831 router, we noted that finished rows are cleaned from the table automatically after about one minute. There are also a couple of response parameters that track the progress of the file transfer, as indicated in Table 6.2. They can be retrieved from the ccCopyTable row after a transfer has been initiated. Again, all parameters have a prepended 'ccCopy'.

The implementation of this approach is further discussed in Section 8.6.4.

Response	Possible Values	Description
State	waiting(1), running(2), successful(3), failed(4)	Information on the progress of configuration file transfers
TimeStarted	*timestamp*	The time this copy started
TimeCompleted	*timestamp*	If copy is terminated, time when it was finished
FailCause	unknown(1), badFileName(2), timeout(3), noMem(4), noConfig(5)	Indicates the reason for failure if `State` is failed

Table 6.2: CISCO-CONFIG-COPY response parameters.

6.2.4 WBEM and WMI

To make management solutions interoperable, a common standard is necessary. The industry organisation Distributed Management Task Force (DMTF) [DMTF, 2007] was founded in 1992 with the goal of defining interoperable management standards. It succeeded in bringing many of the major hardware and operating system manufacturers together. Their main standard is called Web-based enterprise management (WBEM). This is a set of specifications developed to unify the management of distributed computing environments. Windows Management Instrumentation (WMI) is the Microsoft implementation of WBEM.

WBEM is based on the Common Information Model (CIM), which is a conceptual model of network management. The CIM Schema follows an object-oriented design to model network components. It is structured in three layers. The *Core Schema* captures notions that are applicable to all areas of management. *Common Schemas* model notions which are common to particular management areas, but still independent of particular technologies or implementations. There are common schemas for systems, devices, applications, metrics, databases and physical environments and other areas. *Extension Schemas* allow vendors to create extensions for technology- or product-specific environments.

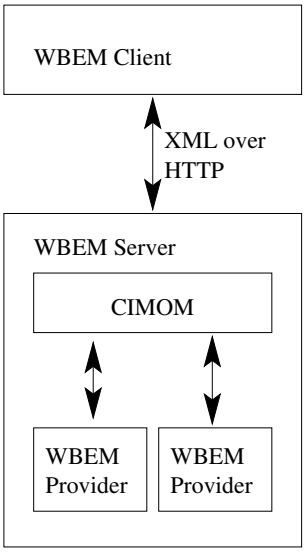


Figure 6.3: WBEM architecture.

WBEM uses HTTP as a transport protocol for method calls expressed in XML [DMTF, 2004]. There is no security concept built into the protocol, but SSL (https) or Secure HTTP can be used. The general architecture is illustrated in Figure 6.3. The machine running the administration software is called client, the device to configure is named server. Exchanging XML messages over a HTTP connection, the client interacts with the CIM Object Manager (CIMOM) on the server. The CIMOM has to interpret the methods and locate the requested *provider*, which will execute the specific commands. To finally retrieve or modify management information, the provider interacts with the system component it is responsible for. This modular structure facilitates the integration of applications from various vendors. WBEM enables the exchange of data across otherwise disparate technologies and platforms.

[Sandlund, 2001] compares WBEM and SNMP on several aspects. He concludes that WBEM is the more complex of the two, as it uses 23 methods, while SNMP only has 9 methods. The BER encoding of SNMP is simpler than the XML encoding used in WBEM. Sandlund identifies slight advantages in security in WBEM, as it permits the use of https or SHTTP. Both offer a broad selection of encryption standards and include the use of server certificates. The main advantage of WBEM is its clean structure. SNMP scatters its information to thousands of MIBs, while WBEM's CIM reaches a single model. The CIM provides a dynamic model, assuring compatibility of classes of equipment. Additionally, WBEM can use mappings from other standards, for example SNMP, to support legacy devices.

Windows Management Instrumentation

Microsoft implemented WBEM in a framework named Windows Management Instrumentation (WMI) [Microsoft, 2003]. Besides providers and the CIMOM, WMI also contains a scripting library for writing management scripts. The scripts not only operate locally, but also on remote Windows machines. WMI uses the Distributed Component Object Model (DCOM), instead of the standard HTTP protocol for communications. According to [Sandlund, 2001], the reason for this is that Microsoft released the implementation before the DMTF decided on a standard transport protocol. WMI has been pre-installed since Windows 2000, but Microsoft offers installation packages for Windows NT / 98 / 95.[3]

The system parts that can be manipulated through a provider are called *managed resources*. The CIMON uses WMI Query Language (WQL), a subset of Structured Query Language (SQL), to select instances of classes. The scripting library permits one to create and delete instances

[3]Look for WMI Core on the Microsoft download website.

of managed resources and to read or modify their properties. [Zurkinden, 2005] reports the same difficulties with WMI as encountered in SNMP: many properties can only be read but not modified. The actual focus of the implementation and of the documentation is on monitoring computers and gathering information about their configurations, not on changing it.

There is a second possibility of manipulating configuration on Windows. Most configuration is stored in the Windows registry database. It would be possible to write a Windows application that receives scripted registry manipulations, a kind of management agent. There are Java libraries that allow to manipulate the Windows registry in Java programs. A C# program would also be acceptable, as the agent has to run on Windows only. However, this would require the administrator to install software on each machine to be managed. Authentication and authorisation would have to be implemented and so on. Even worse, the registry can be tricky to operate directly. It is more stable to use the methods and interaction mechanisms provided by the WMI API. The implementation of the WMI distributor for Verinec is described in Section 8.6.5.

6.3 Related Work in Network Configuration

Network configuration is an area where many applications are available. Some have been mentioned in Section 2.4. There are tools that enable service providers to specifically administer large groups of routers, e.g. from Cisco for their products or the IP Analysis Tools (IPAT) by WANDL, Inc[4]. IPAT is mainly a reporting tool. It permits one to track tasks and monitor routers and can ensure that policies are followed in the productive system. However, configuration changes are supported only rudimentarily. Splat [Abrahamson et al., 2003] specialises on managing only the edge devices. These tools are mostly used by service providers. They do not provide thorough verification of the network configuration. Enterprise or university networks are a lot more heterogeneous than a service providers' infrastructure. Besides switches and routers for the internal network and external connections, one finds various servers as well as workstations that need to be configured.

There are several projects concerned with administrating the computers in university networks. The two most advanced projects in this area have been mentioned in Section 2.4: Cfengine [Burgess, 1995] and LCFG [Anderson and Scobie, 2002]. Their configuration languages are both more generic. This allows for more flexibility in supported services, but lacks possibilities of re-using configuration for devices from other manufacturers. Moreover, the lack of semantic information in the configuration representation makes it difficult to automatically test its correctness. Unlike Verinec, both need an agent running on each device. The Netopeer project [Lhotka and Novotny, 2007] has a configuration language similar to that of Verinec. Parts of the XML Schemas for service configuration in Verinec are inspired by the Netopeer DTD.[5] Netopeer also uses the concept of translation and distribution plugins for the remote configuration of devices. However, this project was developed within the context of an IPv6 project, and is consequently focused on routing, whereas Verinec aims at configuring all network services.

There are management systems available, which use a generic approach to support devices from different vendors. Voyence Control NG [Voyence, 2006] and Intelliden R-Series [Intelliden, 2006] are focused on network devices such as switches and routers. Both support complex network management workflows with different roles and the enforcing of policies for configurations. They also have plugin systems to support new types of devices, and API's to integrate the products into other management solutions. Voyence also has a vendor-independent configuration interface, while Intelliden transforms the proprietary Command Line Interface (CLI) languages into an XML representation defined by an XML Schema. Spectrum [Aprisma, 2006] supports the configuration of both network devices and servers. While supporting multiple vendors, it operates on implementation-specific configuration data. The Verinec approach differs from these commercially available systems in using simulation to test whether requirements for the network configuration are met.

[4] See `www.wandl.com/html/ipat/IPAT_new.cfm`.

[5] Both Verinec and Netopeer are Open Source projects.

Chapter 7

Importing Existing Configuration

The most convenient way of building a reliable network is to start the design from scratch. Unfortunately, this is rarely possible. Most of the time, a management solution will be introduced when the network has grown too large to be administered manually, or when the old solution has proved too inflexible. At this point, the network topology already exists and the network elements have been configured – more or less correctly. Some NMS require the user to define the network configuration from scratch [Bellogini and Santarelli, 2004]. Other projects focus on analysing and monitoring existing networks [Hewlett-Packard, 2007], [IBM, 2007], [Toledo, 2006], but do not have the capability to directly configure network elements. A complete NMS should provide the means of importing existing configurations as well as manipulating and changing the configuration.

Importing an existing network can be divided into two phases. The first consists of gathering information about existing hosts and the network topology by means of traffic analysis and active scans. In this phase, no logins to the machines are done. In the second phase, the exact configuration of the network elements is acquired by means of parsing configuration files, or by similar methods. This time, the application has to do remote logins into each machine in the network. To read all of configuration information, administrator rights are usually required. While the results of the first phase are neither exact nor complete, the second phase yields precise and reliable information.

In some situations, it can make sense to apply only one of the phases. If the network topology is already known, the exact configurations can be retrieved for single network elements with known names. On the other hand, if only the network layout is of interest, there is no need to import the exact configuration. If both phases are to be passed, the obvious order is to first detect the existing devices and the network layout and to then import configuration details from the network elements.

7.1 Detecting Existing Network Layout

Detecting network elements within an existing network can be accomplished by using traffic analysis. By looking at the type of application protocol employed, it is also possible to guess what services are run on which ports. Range scans can be used to detect running but inactive hosts. To determine the network topology, traceroute methods can be used to identify the routers between the management station and the initially detected hosts. Network elements found by traffic analysis and tracing can be further examined using a portscanner.

Analysing the network setup does not need administrator access to the machines.[1] However, some of the techniques discussed in this section require permission from the network staff. In particular, the process of scanning a device for open ports looks like the preparation for an attack on the network. Intercepting traffic might also collide with the network usage guidelines. Thus, experiments are best done in a closed experimental network. One should at least adhere to all relevant policies used in the organisation and ask for all required permissions.

[1] A privileged account is usually required on the machine running the traffic analyser. Many applications need raw access to the network interface, which is forbidden for normal users.

7.1.1 Traffic Analysis

A lot of information about a network can be gathered by looking at network traffic. During normal operation, the operating system passes only packets concerning the local machine to the applications, and discards all other packets. When doing *packet capturing*, one is interested in all packets seen by the network interface, even those not addressed to this particular interface. It is possible to open the interface in the so-called "promiscuous mode". In this mode, it will report all packets that are detected, regardless of their destination address. Most operating systems require administrator privileges for this level of access to the interface driver.

Traffic analysis, also called "Sniffing", can be used both for good and ill. Legitimate uses for traffic analysis are, for example, troubleshooting and analysis of the network, debugging of network protocols and education. On the other hand, it can be used to glimpse confidential information passing over a network. Several unprotected protocols are still in use to date, such as HTTP, SMTP, POP3, FTP or Telnet. The contents of these protocols are not encrypted, and thus all information can be read from captured packets. Protocols like POP3 or FTP also send usernames and passwords in clear text, which makes it possible to discover them by using a traffic analyser tool. Data sent over the Internet is observable at all machines it passes. Legitimate traffic analysis is most often encountered in the context of examining network protocols. The most famous traffic analyser is Wireshark [Combs et al., 2006], available for several operating systems.[2] There are other tools, such as the commercially available [Agilent, 2006], [CommView, 2006] or Microsoft's Network Monitor which is part of the SMS Server, and the open source programs [Jacobson et al., 2006], [Ornaghi and Valleri, 2006] or [Song, 2001]. Wireshark provides a sophisticated user interface to define what to capture and to filter the results. It understands the protocols and can relate packets from the same communication exchange to provide logs for specific communication interactions. The information contained in headers as well as the protocol payload are presented in a readable way. Wireshark itself has no implementation for capturing the packets, but uses the libpcap library [Jacobson et al., 2006]. For integrating traffic analysis into a network management application, Graphical User Interface (GUI) applications are less suited. Using a library like libpcap directly allows one to specify more precisely what information should be gathered.

Detecting Network Elements

For network elements detection, the traffic analysis is undertaken with a different intention than in Wireshark. We are not concerned with protocol details, but with the host names and port numbers the packets reveal. Every TCP or UDP packet contains a header naming its source and destination host IP address, as well as the port number. By looking at the connection bits of TCP, it is also possible to tell which host is initiating a connection, and to distinguish clients and servers. The packet body usually contains protocol payload, but in determining the hosts, the payload is irrelevant. The packets are only used to determine which network elements exist and which services seem to be running on them.

The result of the traffic analysis phase is a set of IP addresses and open ports on each address. The DNS system can be used to look up the names corresponding to the IP addresses, a function called *reverse DNS*. For reasons of efficiency, the name lookup should be done later, once the topology has been searched out.

There are several tools for network elements detection based on traffic analysis, for example EtherApe [Toledo, 2006]. However, their sole purpose is displaying the network layout. For Verinec, a custom implementation has been developed, based on libpcap.

Switched Networks

When hubs are used to connect the network, all of the traffic in the network segment arrives at the network interface and can be analysed. In switched networks, packet sniffing is more difficult. The switch usually creates direct connections between the communicating machines, thus reducing the segment size to the machine and its communication partners. The interface of the sniffing host only sees traffic destined at or originating from that host. Enterprise switches sometimes provide the ability to mirror all of the traffic to one port. This is called *Port Mirroring*, *Monitoring Port*

[2] Wireshark has been known as Ethereal, but was renamed due to trademark issues.

or SPAN (Switch Port ANalyzer) and for Cisco devices [Cisco Systems, 2007a]. If such a device is to be used in the network, the traffic analyser has to be connected to the mirroring port.

If the switch does not have a monitoring port, it is difficult to find any network elements that are not communicating with the sniffing machine itself. The dsniff toolchain [Song, 2001] contains a utility application called `arpredirect` to circumvent the restrictions of switched networks. It sends forged Address Resolution Protocol (ARP) packets to other machines in the network, telling them that the attacker system is the new default gateway for all other devices. This attack is called "ARP spoofing". The other hosts will start sending all traffic to the attacking system, assuming it to be the gateway. The attacking system forwards the packets to the real gateway, to avoid interrupting the communication. As the traffic is now routed over the attacking system, packets can be analysed again.

Another attack called "MAC flooding" can cause switches to fall back to the "hub-like" behaviour of forwarding received packets to all ports. This attack consists of sending the switch large amounts of bogus MAC address data.[3] The switch has a limited memory for the translation table, telling which MAC address is connected at what port. When the table is full, the switch has to fall back to broadcasting the packets, as it does not know on which port to send them out.

However, attacking a productive network on that level might not be a good idea. Service quality can decrease or be entirely interrupted. As NMS are intended for keeping important networks running, such risks should not be taken. Additionally, traffic resulting from the attack might also interfere with the network analysis. If the switch has no mirror port, the safe approach is to make the machine running the sniffer communicate with as many of the servers as possible.

Range Scans

Even if all switch traffic is seen at the interface, inactive hosts that neither send nor receive packets during the sniffing period remain invisible. Traffic analysis is a passive method, seeing only what arrives at the network interface. To increase the number of findings, active methods can be used. One promising method is scanning a whole IP range for responses by hosts, sometimes called *sweeping* [Skoudis and Liston, 2006, 262]. The method consists of determining an appropriate range of IP addresses, here usually the subnet the NMS will be used for, and sending a packet to each address. Skoudis and Liston list three types of interesting packets: ICMP echo (aka Ping), TCP connection request on port 80 or UDP requests to probably unused ports. ICMP echo packets should be replied to by all hosts, but they are sometimes turned off, or else firewalls might filter them because the network administrator wants to prevent scans of the network. TCP connections are rarely filtered, but not all hosts can be expected to run a server on port 80. The UDP method relies on the machine sending a ICMP "port unreachable" response, which nonetheless reveals the existence of a machine at that IP.

To determine what IP ranges should be scanned, the information previously gathered by traffic analysis can be used. Existing IPs indicate the subnets used in the local network.

Limits

The list of network elements that can be detected by traffic analysis and range scans is by no means complete. It is not possible to tell how many of the existing network elements are detected. The packet capturing process only sees packets passing at the network interface of the machine it is running on. Services not used during the sniffing phase are not found. Intermediary network elements, typically routers, also remain undetected. The routers might be revealed using traceroute techniques, as described in the next subsection. It lies in the nature of the analysis that inactive network elements cannot be found. Range scans can be hindered by firewalls or host configurations. If a device is switched off, it will not respond to any requests, effectively hiding it even from range scans. The administrator of the network will have to check the configuration and add missing elements by hand.

Sometimes, traffic analysis might also report wrong network elements. The packet capturing is based on IP addresses. A single network interface can impersonate more than one IP address, and a host can contain more than one network interface. Every distinct IP address will be recorded as

[3] In a reasonably well protected network, intrusion detection systems should be able to detect and prevent this type of attack.

a complete network element in the first run. If, in the second phase of the import process, parsing interface configuration data on some host reveals that different IPs belong to the same host, their configurations have to be merged into one host with several interfaces.

Another problem arises when several names map to the same host, for example to implement name-based virtual hosts with HTTP. To achieve virtual hosting, DNS reports the same IP for different host names. The web server reads the actual host name to use in an HTTP request header field. When looking up the host name for an IP using reverse DNS, there can be only one answer, which will be the principal name for the host. Theoretically, it is possible to examine the HTTP requests telling the host names. However, the information thus gathered would probably remain incomplete and the list of names is best retrieved by importing the DNS configuration data, as described in Section 7.2.

7.1.2 Network Topology Search

In December 1988, Van Jacobson wrote the first *traceroute* program [Leinen, 2006]. The intention was to see which devices transmit a packet destined for a certain network address. To determine the nodes, packets with increasing Time To Live (TTL) values are sent, beginning with 1. The routers on the way send back ICMP "TTL exceeded" messages, which carry their IP as sender address. Because RFC 792 specified that "TTL exceeded" messages need not be sent for timed out ICMP packets, the original implementation used UDP packets. RFC 1122 later required all Internet hosts to send "TTL exceeded" messages for ICMP packets as well. The Windows implementation of traceroute always sends ICMP packets, Unix/Linux implementations often allow a choice between UDP or ICMP. There are also implementations using TCP, for Unix systems `tcptraceroute` [Toren, 2006] can be used, for Windows there is `tracetcp` [Witek, 2005].

Unfortunately, not all machines send TTL exceeded messages, and some firewalls even drop the messages to protect the network from such analysis. While this can improve security and performance, it provides a challenge for traceroute implementations. If just one machine in the chain does not respond, but the next does, the target host has not yet been reached. However, if the target does not respond it is impossible to determine whether increasing the TTL will lead to results or not. The sender cannot tell whether the target or an intermediary host is not responding. There are thus several decisions to take when designing a traceroute tool. How long should the tool wait for responses before it times out? How many times should the packet be resent after a timeout to check whether a packet got lost in the network, or whether there is really no response? And finally, after how many TTL increases that resulted in timeouts should the tool assume that it has reached the target, but that this host is not responding?

Typical implementations wait for a couple of seconds before assuming a timeout and then retry three times. They accept up to 10 non-responding hops before they give up. If a target host does not respond, this can lead to rather long running times.

Traces help discovering tree structures that have the scanning host as a root. Additional connections between routers that exist in order to increase capacity or to provide fallback in cases of line interruptions will not be found. This information about the network structure could be gathered by reading the routing tables of the routers.

7.1.3 Port Scans

Traffic analysis reveals devices in the network. It can detect which protocols are used between them to guess what services they offer. Additional information about a detected device can be gained by looking at the reaction when sending packets to different TCP or UDP ports. This technique is called *port scanning* [Fyodor, 1997] and helps to determine what services a host is offering. Tools like Nmap [Fyodor, 2006] try to connect to a set of ports of a device, and can optionally test whether it is running the usual service for well-known ports[4] or else, determine which service it is. By analysing details in the protocol behaviour, the so called *finger prints*, Nmap is even capable of identifying the software implementing the service right down to its version number [Fyodor, 1998]. Again, there are other tools for port scanning, for example [angryziber, 2006] or

[4]For common application protocols, a mapping of protocol to default port number exists. HTTP has port 80 for example, ssh port 22, and so on. Hoping for "security by obscurity", administrators sometimes run services on different ports.

[Lee and Louis, 2006]. Port scanners are mostly used either to prepare an attack on a host or by security staff to determine whether a host is vulnerable. Tools like Nessus go one step further, checking the found ports for known vulnerabilities and directly telling the user which tool could be used to break into the host.

Detecting Services

When gathering information about the existing network for an NMS, vulnerabilities of the hosts are not the main concern.[5] The only concern is what service runs on which port. For each protocol that has to be detected, special packets must be forged and patterns have to be found. While it is possible to implement protocol detection mechanisms using a low level socket library, using an existing port scanner is probably a better idea.

One could guess at configuration details of services from the outside by examining the reactions to various requests. However, this would never be as accurate as importing the exact configuration using an administrator access, as described in the next section.

7.2 Configuration Import

The previous section discussed how to detect network components. A lot of information can be gathered about a host by scanning it via the network. However, in this second phase of network import, we want to gather precise information about each network element. With the results of the port scan and an administrator providing the necessary authentication details, the analysis of the network elements can continue. Possible methods of gathering information include SNMP, parsing configuration files on Unix hosts, executing programs via remote shells and analysing their output. In this section, some examples of how services can be imported are presented.

7.2.1 Hostname

The name of a device can sometimes be found during the scanning process or be retrieved using reverse DNS. When importing the settings of a local Unix machine, the command `hostname` can be used to determine the name. If the configuration is imported from a remote device, a DNS hostname might specified by the user as one of the connection paramters, if the user does not provide the numerical IP. The DNS name does not necessarily coincide with the hostname the machine is using internally. If local copies of configuration files are imported, the name must be specified by the user.

7.2.2 Network Interfaces

As an example of network interface configuration, we examined the Fedora Core Linux distribution by Red Hat. To set up the interfaces at boot time, the Linux system stores configuration in files. Red Hat distributions [RedHat, 2003] have one file for network information, called `/etc/sysconfig/network`, and a file for each interface in `/etc/sysconfig/network-scripts/`. The interface files are named `ifcfg-`*name*, with *name* being the name for the interface, for example `eth1` or `wlan0`. The file names have no meaning for the configuration, it is merely a naming convention that makes it possible to easily find the corresponding file for an interface. The files are formatted in the quite simple "ini-style" a list of *name* = *value* pairs. Other distributions may vary in the location and format of the configuration files, although not in substantial matters. Some use one single file for all interfaces, for example Debian and Ubuntu, OpenBSD uses different file names and FreeBSD includes the configuration in its `/etc/rc.conf` file.

The Fedora Core files do not use advanced ini-file features such as sections. They simply contain parameter mappings and comment lines which are to be ignored. Figure 7.1 shows a listing of a typical file to configure the first Ethernet interface.

There are two kind of files that need special treatement. The file `ifcfg-lo` configures the loopback interface present on every Unix-like system. The loopback device is a virtual network

[5] Although including warnings for security issues detected during the scan would enrich the possibilities of the NMS as a vulnerability assessment tool.

```
# ifcfg-eth0
DEVICE=eth0
BOOTPROTO=dhcp
HWADDR=00:06:5B:A9:EF:60
ONBOOT=yes
TYPE=Ethernet
```

Figure 7.1: Typical contents of the file ifcfg-eth0.

interface, connecting to the local machine itself. It is used whenever networked client and server applications run on the same machine. Normally, an administrator does not need to do anything with the loopback configuration. It is created on system installation and no changes are required. The other special kind of configuration files define *alias* interfaces. An alias for an interface is an additional IP address that the physical device should listen to. The files are named `ifcfg-`*name*`:`*alias*, for example `ifcfg-eth0:0`. Alias files do not define an interface in its own right, but are part of another interface. This must be correctly reflected in the imported configuration, using multiple `nw` elements, as defined by the node XML Schema.

The interface configuration information could also be retrieved from the output of the command `ifconfig -a`. While the output would be the same across different Linux distributions, it is less convenient to parse. It is formatted for human readers and contains usage statistics. More seriously, it is neither possible to tell from the output whether the IP is configured statically or by DHCP, nor is it visible whether the interface is to be brought up automatically at boot or not. Therefore, the results of parsing the configuration files are more precise. Unfortunately, this means that different implementations are needed to support the major Linux distribution families.

7.2.3 Packet-Filtering Firewall

Importing packet filtering configuration is discussed on the example of the Netfilter framework found in most current Linux systems [Welte et al., 2006]. A general introduction to the operation of Netfilter has been given in Section 3.4 of this thesis. Retrieving the configuration information for Netfilter is not a trivial matter. Red Hat Linux stores the configuration in the `/etc/sysconfig/iptables` file. The format of this file is a list of arguments to the `iptables` command, which is the front end used to configure Netfilter. Other Linux distributions use other formats and store the file under different names.

The call to iptables with the arguments `/sbin/iptables -vnL` prints a list of all currently active rules in all chains of the filter table. The output of this command is the same on all systems using Netfilter. An example of the output is shown in Listing 9. The rules drop all packets coming from or being sent to a host with IP 192.168.199.3.

Listing 9: Output of iptables.

```
buchmand@diufpc230:~> sudo /sbin/iptables -vnL
Chain INPUT (policy ACCEPT 12791 packets, 6610K bytes)
 pkts bytes target  prot opt in  out  source          destination
    0     0 DROP    all  --  *   *    192.168.199.3   0.0.0.0/0

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target  prot opt in  out  source          destination
    0     0 DROP    all  --  *   *    192.168.199.3   0.0.0.0/0
    0     0 DROP    all  --  *   *    0.0.0.0/0       192.168.199.3

Chain OUTPUT (policy ACCEPT 13658 packets, 1811K bytes)
 pkts bytes target  prot opt in  out  source          destination
    0     0 DROP    all  --  *   *    0.0.0.0/0       192.168.199.3
```

Unfortunately, iptables cannot output the configuration in XML format. The output format is intended for human reading rather than for programs. The syntax is not documented. The information provided here and used in the Verinec application was reverse-engineered from the man pages, and by executing iptables with all possible settings. [Antener, 2005] documents the complete grammar for the output of iptables. This was established with iptables version 1.2.9, other versions might use different formats. There was no parser to be found for the format. To implement the import, a custom parser has been developed by [Antener, 2005].

Each chain has a header line, defining its name and the default policy, along with some statistics. A chain can have zero, one or more rule lines. If there are rules in the chain, they appear after a label line with descriptions for nine columns. The actual rules have one additional, unlabelled column for options. Cells with no values are indicated by the character '*', which allows columns to be counted. The columns of the iptables output are:

pkts: Statistics of number of packets handled. Irrelevant for configuration import.

bytes: Statistics of traffic volume handled. Irrelevant for configuration import.

target: Target or chain to jump to if rule criteria match.

prot: Required packet protocol type for the rule to be matched (tcp, udp, icmp, ...)

opt: Option describing whether the packet is fragmented or not.

in: Interface that must have received the packet for the rule to matched.

out: Interface the packet would normally be sent out from for the rule to be matched.

source: Source IP address of the packet (0.0.0.0/0 matches all addresses).

destination: Target IP address of the packet (0.0.0.0/0 matches all addresses).

options: All other criteria of the Netfilter framework.

The iptables output contains all information required to determine the configuration. The implementation of Netfilter import for Verinec is discussed in Section 8.8.2.

7.2.4 BIND Domain Name System

As explained in Section 3.5, the Domain Name System (DNS) is used to match IP addresses and host names. One important implementation of a DNS server in the Unix and Linux world is BIND (Berkeley Internet Name Domain) [ISC, 2007]. This section holds some notes about how an importer for BIND could be implemented.

The main BIND configuration file is located under `/etc/named.conf`, where global options such as paths are set and the zones are declared. Other features of BIND include the definition of access control lists to specifically allow or deny requests coming from certain IP addresses. The format of zone entries in the main file is shown in Listing 10, along with an example. A parser has to read the zones and handle the global options to locate the correct files with zone definitions.

Listing 10: Zone declaration in named.conf

```
/* format:
zone <zone-name> <zone-class> {
     <zone-options>;
     [<zone-options>; ...]
};
*/

zone "unifr.ch" IN {
  type master;
  file "unifr.ch.zone";
  allow-update { none; };
};
```

The actual zones are stored in individual files, one for each zone. Listing 11 shows a typical zone definition. The main file tells the name of the zone for the `match` and the `type` attribute.

Listing 11: Example zone file unifr.ch.zone

```
; zone file for unifr.ch
@    IN   SOA  siuftesun11.unifr.ch. hostmaster.unifr.ch. (
              2007040800 ; serial
              12h        ; refresh
              15m        ; retry
              3w         ; expire
              3h         ; min_ttl
              )
              IN  NS     siuftesun11.unifr.ch.
              IN  MX 10  mail.unifr.ch.
siufsrv81     IN  A      134.21.201.101
siuftesun11   IN  A      134.21.201.200
www           IN  CNAME  siufsrv81
```

All remaining information is contained in the zone file. The first line states the authority `primaryns` for the domain and the `adminmail` in hostname notation (the at character '@' is replaced with a dot). The meaning of the numbers in the header section is explained in the comments in Figure 12. The header section is followed by a list of entries for actual name to IP mappings and other DNS information. The entries have the format `match [ttl] class type [type-parameters]`. The most important record types are listed here, a complete reference is found in [Aitchison, 2007].

match: The host name. If it is not an absolute name with a dot at the end, the domain of the zone is appended.

ttl: Optional entry to overwrite the default time to live from the start of authority header.

class: In this context always `IN`, for Internet protocol.

type: Resource record type. Important are:

NS: Authoritative name server for the domain.

MX: Mail exchange server. The parameters specify the priority of the mail server.

A: Entry for a hostname to IP mapping.

CNAME: Canonical name. Defines an alias name for a host.

TXT: Additional text information associated with the domain.

type-parameters: Some types have additional parameters.

A generic method to import DNS configuration would be to use the DNS protocol to start a *zone transfer*. For a zone transfer, a machine asks the DNS server responsible for a particular zone to transmit all zone data to it. Verinec could implement this part of the DNS protocol and ask the DNS servers for their zones.

Currently, there is no implementation for an importer of DNS configuration in Verinec. When implementing one, the first decision to make is whether to import DNS data over a zone transfer or by parsing configuration files of different implementations. This will depend on whether the information received in a zone transfer is as rich as that in the implementation-specific configuration files.

Part III

Verinec Prototype Implementation

Chapter 8

Verinec Implementation

The Verinec application has been designed with the different aspects of network management in mind discussed in the previous chapters. The main principle of Verinec is: configuration has to be verified before it is applied to devices. This implies not only test cases for the configuration. Proper distribution of configuration is also important. Validated configuration should never have to be manipulated "by hand", but handled automatically by the NMS, as suggested in Chapter 6. The focus of Verinec lies on one class of network service interruption: misconfiguration. General discussion of misconfiguration and its implications are found in Section 2.3.

The architecture of the framework is joint work from the author of this thesis and from Dominik Jungo. The adaptation part and the configuration XML Schema has been implemented by the author, the verification part by Dominik Jungo. Different tasks where done in the form of Master and Bachelor theses at the University of Fribourg, under the supervision of the author. The authors of these theses are credited in the corresponding sections.

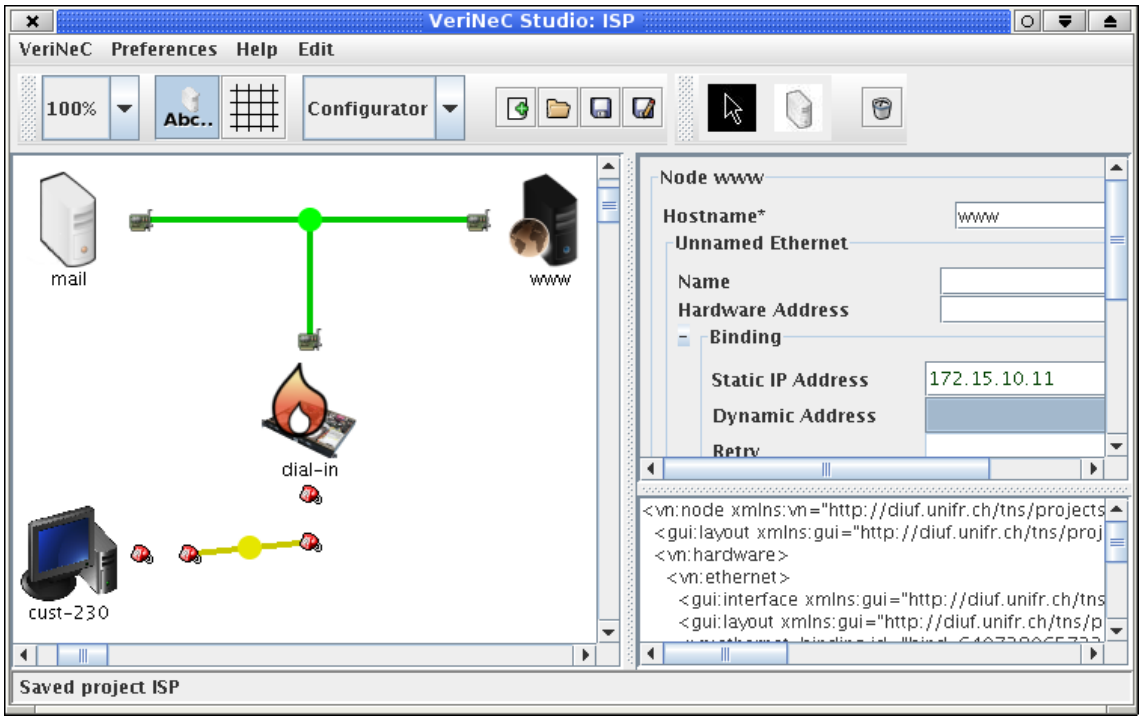


Figure 8.1: Screenshot of the Verinec GUI.

This chapter begins with explaining the architecture of Verinec. Then, the Verinec configuration data format will be discussed. Network layout and node configuration are expressed in XML, according to an XML Schema definition. Several modules are built around the network definition.

Each of the modules has a section in this chapter. To configure the real network, the adaptation module can automatically configure devices. There is an editor module to define and edit the network configuration. The verification module allows to check the configuration and simulate the network to test its behaviour. To create a network configuration from an existing network, the import module is used. It comprises a network scanner to automatically discover the layout of the network and a parser for configuration data of nodes. Chapter 9 illustrates the usage of Verinec from the point of view of a user (network administrator) on a real example.

8.1 Architecture of Verinec

Verinec is built around an abstract network definition. Its main elements are "nodes" (network elements: computers, routers, appliances, etc.) and network connections between the node's network interfaces. Each node can have configuration for services. The format will be described in Section 8.2, service configuration in Section 8.3. Named collections of nodes and network connections can be saved and loaded. The main application consists of a GUI to graphically visualise the network layout and a framework to work with the network elements. Figure 8.1 shows a screenshot of the application.

The work flow of the Verinec application is illustrated in Figure 8.2. Configuration from the real network can be imported using several methods. Once the abstract configuration of the network exists, it can be verified using rules and simulation. This provides feedback for an administrator to fix the errors in the editor. To deploy the new configuration to the real network, it is translated and distributed using an appropriate method. The automated configuration distribution has two advantages over manual configuration after the design process. Manual configuration is tedious for the administrator and a waste of time. The second and more important advantage is the security gain. Correctness verification would loose its significance if errors could be introduced in the process of configuring devices. Problems with the translation also cause feedback for the editor, for example when the abstract configuration uses features not supported by the target implementation. An added value of Verinec is the availability of a complete network description.

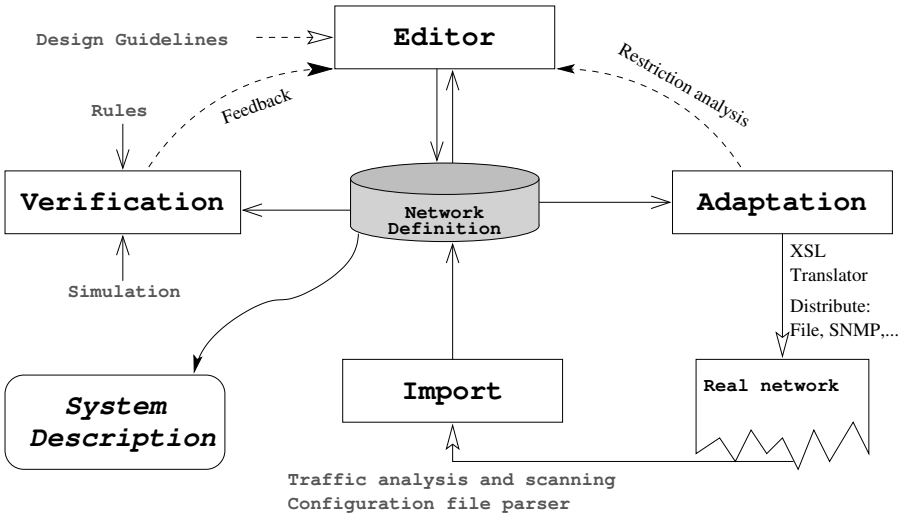


Figure 8.2: The work flow of the Verinec application.

The Verinec application is built in a modular way. Various modules can be plugged into the GUI. The existing modules are introduced in this section and discussed in depth in Sections 8.4 to 8.8. Section 8.1.1 explains the module framework in order to extend Verinec with own modules. Available modules are:

Editor: Provide the functionality to alter node configuration and network layout.

Adaptation: Translate the abstract XML configuration into implementation-specific format and distribute it.

Verification: Test if a network fulfils specified requirements, using the simulator and rules on the XML documents. The network behaviour is simulated using the XML definition. Besides verifying the configuration, this module can also visualise the network simulation.

Import: Analyse network traffic and import configuration files to create abstract XML data for an existing network.

The graphical **editor** allows to edit the network. One can add and remove machines, interfaces and network connections. Configuration of Ethernet interfaces is supported with a graphical editor written by [Vogel, 2005]. Configuration of other services is not yet supported by graphical editors and must be done in source code. The usage of the editor from a network administrator side of view is explained in Chapter 9.

Verinec is more than just a network design tool. It is also capable of deploying the configuration to the target devices on the real network using the **adaptation module**. The configuration is transformed into an appropriate format and deployed by means of copying configuration files or using remote administration protocols. The adaptation module is discussed in Section 8.4.

To improve network reliability and security, Verinec incorporates a **verification module**. This module analyses the abstract configuration and contains a simulator to test the behaviour of the network. Requirement definitions allow to automatically test for the required network functionality. Using the testing module can reveal configurations errors before the setup of the productive devices is actually altered. Section 8.7 further explains the verification module.

For practical use, Verinec contains an **import module** to detect existing system setup and generate the abstract configuration. It uses traffic analysis to detect the network layout and parsers to import configuration files, use SNMP or similar methods. The vision is that an administrator can run Verinec and import her existing configuration and then deploy it on another machine running a different operating system and this other machine will behave exactly the same way the first one did. The import module is covered in Section 8.8.

8.1.1 Fundamental Code Design

This section provides an introduction to the Java code implementing the Verinec application. For a full reference, see the javadoc source code documentation. The GUI had initially been implemented by [Loeffel, 2004], but was extended and refactored many times.

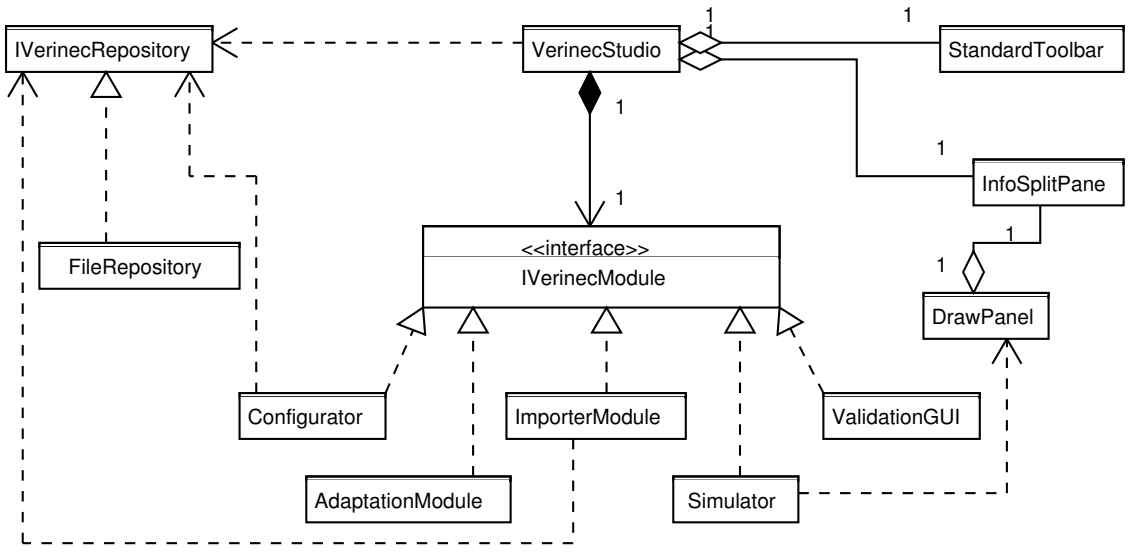


Figure 8.3: The main Verinec framework.

Figure 8.3 describes the core classes of Verinec. The singleton[1] `VerinecStudio` is the main application window, extending `JFrame`. At the same time, it serves as a reference point for the different modules. The visible window is divided vertically by the `InfoSplitPane`, as illustrated in Figure 8.1. In its left side, the `DrawPanel` is embedded to display the network layout and nodes in a graphical representation. Its right side is used for module specific editors. The `Configurator` module plugs in a configuration editor, while the `Simulator` module adds an editor for simulation input events. If there is no specific editor, the configuration of the currently selected element is shown in XML source. Following standard design principles, menus and tool bars are placed above the main area. `VerinecStudio` provides a menu to save and load networks and one to configure the application and modules. The standard tool bar provides icons for the save and load operations. Additionally, it allows to switch between the modules, hide or show node names and the layout grid and to set the zoom level. The modules may add their own menus and tool bars.

All network configuration is stored in an `IVerinecRepository`. The repository interface defines methods to store and load networks and nodes. Implementations can use the file system, XML databases or any other suitable method to store the data. Verinec currently contains the `FileRepository` implementation, which stores nodes as files on disk.

The various tasks of Verinec are implementations of the interface `IVerinecModule`. Figure 8.4 shows more details on the modules, namely the methods of the `IVerinecModule` interface and the relevant methods of `VerinecStudio`. This section explains only the basic concept, the exact details for writing additional modules can be found in the source code documentation.

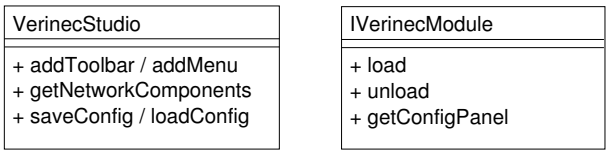


Figure 8.4: Classes and methods for the module framework.

`VerinecStudio` is used as pivotal point for the interaction between the modules and the GUI elements. When the module is loaded, its `load` method is called. This gives the module the opportunity to attach its own icon bars (`addToolbar`) and menus (`addMenu`) to the graphical interface. It can also add module specific commands to single nodes or interfaces by iterating through the network components. This are the only means for a module to make its functions available. The `saveConfig` and `loadConfig` methods have nothing to do with network configuration, but allow the module to save customisation settings before being unloaded and reload them when being selected again. Modules can retrieve the repository from `VerinecStudio`. To modify configuration, the network components framework described in the next section has to be used.

When the module is unloaded, VerinecStudio takes care of cleaning up any menu entries the module might have added. It simply reinitialises all entries to start with a clean state for the new module. Nonetheless, `VerinecStudio` calls the module's `unload` method, so that it can clean up or store state information if needed. Finally, the editor can ask the module for a configuration dialog. If there is nothing to customise with that module, the method may return `null`. If it returns a `JPanel`, that panel is included in the configuration dialog.

8.1.2 Network Components

The UML diagram in Figure 8.5 shows the network components. The `DrawPanel` is the area containing the network. The network consists of components and wires. Components are `NwInterface`, `NwBinding` and `NwNode`. The nodes are further divided into `NwHub` and `PCNode`. A `NwHub` models an unmanaged hub that forwards packets received over one wire to all other wires. Every other device contains some logic and is a PCNode. A hub is automatically created when two bindings are connected together. A `NwWire` always connects a binding with a hub. To add more bindings to the same collision domain, the bindings have to be connected to the hub. The bindings are

[1] A singleton is a class which may not have more than one instance. As this class represents the main window, we do not want it to exist more than once. For more information, see [Gamma et al., 1995]

contained in interfaces, which are in turn contained in the PCNodes. Actual network types are not hard coded in Verinec. The only distinction is made between protocols allowing for more than one binding per interface and those limiting the number of bindings to one.

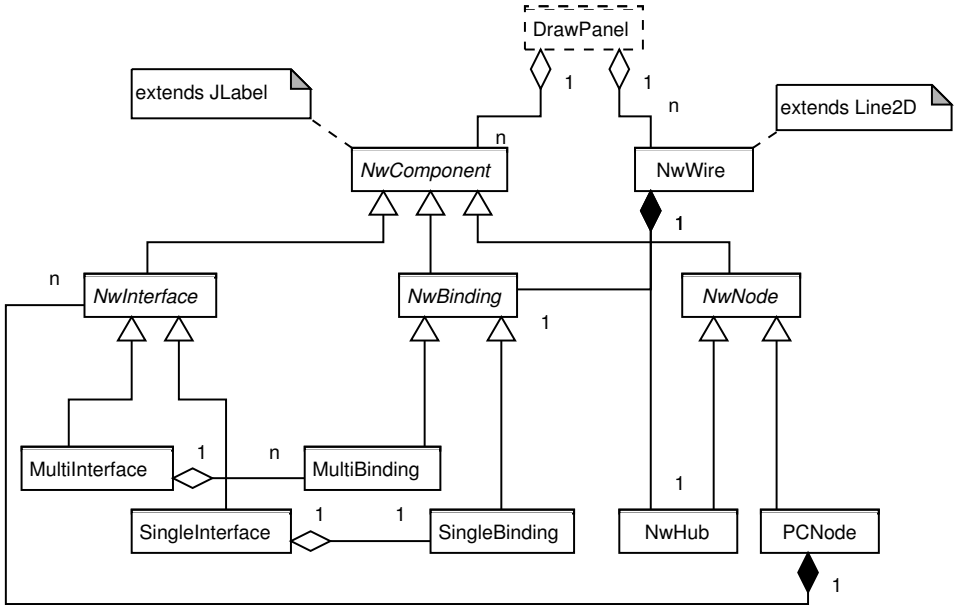


Figure 8.5: Classes to model the network.

The network types are defined in an XML document. Listing 12 shows the default type declaration. The type declaration maps the interface types to their XML representation and declares how to display them. Colours, wire style and icons can be adapted to user preferences in the Verinec GUI. Even though wireless network technologies have by definition no wires, they are handled by the NwWire class, as it is conceptually equivalent. Hubs and bindings can only be connected if they are of the same type.

Listing 12: NetworkTypes.xml

```
<networktypes xmlns="http://diuf.../verinec/network-types">
  <type typeCode="100" guiName="Ethernet" xmlName="ethernet"
        multiBinding="false"
        interfaceIcon="/res/pictures/ethernet.png"
        bindingIcon="/res/pictures/ethernet.png"
        hubIcon="/res/pictures/EthernetHub.png" wireStroke="solid">
    <RGB r="0" g="200" b="0"/>
  </type>
  <type typeCode="200" guiName="Wireless" xmlName="wlan"
        multiBinding="true"
        interfaceIcon="/res/pictures/wireless.png"
        bindingIcon="/res/pictures/wireless.png"
        hubIcon="/res/pictures/WirelessHub.png" wireStroke="dashed">
    <RGB r="0" g="0" b="200"/>
  </type>
</networktypes>
```

The concept might let one think that new network types could be added. This is not exactly true, as the GUI will need to map the network types to XML elements, and for this they must be defined in the node XML Schema.

For a module to create a new node, the method `VerinecStudio.addNode` has to be used, because simply adding the XML definition to the configuration does not build the GUI elements. Where available, specific methods of the `PCNode` like `createInterface` should be used. For configuration not having such methods, `NwComponent.getConfig` can be used to get the XML fragment that defines this component. If the XML configuration is directly modified, `VerinecStudio.nwComponentStateChanged` has to be called, otherwise the GUI won't be updated. To connect two interfaces, the method `GuiUtil.connectNetwork` has to be used, automatically creating the necessary NwHub between the interfaces.

8.2 Configuration Definition Schema

A crucial base of every NMS is its configuration representation. An NMS can store configuration in implementation-specific formats. Such an NMS may provide backups of old configuration, check the syntax to avoid invalid configurations and might enforce policies. Adding support for new services involves little work, especially if the NMS lets the administrator simply edit configuration text files in the original format. However, it is not adapted to support different implementations of the same service. It cannot help translating configuration from a device to one with another implementation. Either the administrator has to know each implementation-specific configuration file syntax or editors have to be developed for every implementation of each service. Other NMS use simple schemata like "name=value" [Anderson and Scobie, 2002] or scripting languages [Burgess, 1995] to store configuration. This may help to adapt to different implementations, but does not support correctness testing of the semantics of the configuration.

For network verification, a semantically rich format is important, as the application has to understand the meaning of the configuration data. The Verinec configuration format is designed to completely specify the semantics of the configuration and is thus rather complex. Verinec configuration is expressed in XML. Every device in Verinec is uniformly considered a *node*, whether it is a workstation, server machine or specialised device. The representation of nodes and their services is defined in the XML Schema `node.xsd`[2]. Each node definition contains a 'hardware' section for its interface configuration and a 'services' section, where all the services provided by this node are configured.

For each service, there is a configuration format specification. The formats are abstracted from actual implementations. By studying the different implementations for a service, a XML Schema has been deduced that covers the important features. All implementations of a service base on the same concept or standard. Although the configuration languages can vary considerably, the number of possible options that must be configured remains limited. Defining the standard features was quite successful for the examples implemented in the Verinec project. Some implementations do not support all features from the abstract configuration. Sometimes, features can be emulated using clever translations. Where this is not possible, the adaptation module warns the user about the problem.

Packet filtering firewalls are a good example to illustrate the power of abstracted configuration. All implementations basically do the same. They examine IP packets received on the different ports of a node and decide whether to accept or reject it, based on source and destination address and various other properties of the packet. There exist different implementations of firewalls both in software and as specialised hardware components. Some of them are configured using configuration files with varying syntax, others use SNMP or command line interfaces. The abstract Verinec XML configuration allows to build structured firewall rules. The translation module allows to use the same abstract configuration for a Cisco appliance as well as for a Linux computer with the Netfilter framework.

The other aspect the configuration has to define is how the nodes are connected to each other. The network layout is defined by defining which interfaces of the nodes are connected together. Those connections are stored according to the XML Schema `network.xsd`.

[2]The full namespaces for Verinec Schemas are `http://diuf.unifr.ch/tns/projects/verinec/<name>`

8.2.1 Nodes

Every network element is uniformly considered as a node, be it a switch, a desktop computer, a hardware firewall appliance or a server machine. Each node has a name attribute and two XML sections: the hardware section for its network interfaces and the services part. An example node definition can be studied in Listing 13. The `hardware` part defines all interfaces this node has. For an introduction to the interface settings, see Section 3.1. This section explains how interface settings are modelled in Verinec. The `services` part is a container for the configuration of the various services that may run on a node. The formats of the service configurations are discussed in Section 8.3.

Listing 13: A sample node

```
<nodes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://diuf.unifr.ch/tns/projects/verinec/node
      http://diuf.unifr.ch/tns/projects/verinec/node.xsd"
  xmlns="http://diuf.unifr.ch/tns/projects/verinec/node">

 <node hostname="pc01">
   <hardware>
     <ethernet>
       <ethernet-binding id='ebna3zt'>
         <nw id='i10' address='192.168.0.55' type='ip'/>
       </ethernet-binding>
     </ethernet>
   </hardware>
   <services>
     <dns>
       <Zone match="unifr.ch" type="master"
          primaryns="ns.unifr.ch"
          refresh="10800" retry="3600"
          ...
       </Zone>
     </dns>
   </services>
 </node>
</nodes>
```

Network interfaces are another good example for how abstract configuration can properly reflect the possibilities of the reality. Every interface in an IP network is either statically assigned an IP number or uses DHCP to receive configuration from a server. Although the names for interfaces and configuration formats vary, this is the same for all operating systems. Linux typically enumerates the cards as ethX and uses files to set the properties. In Windows, the registry key `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\Interfaces` is used. Verinec defines the basic properties like DHCP client or static IP, network mask and gateway. The information is only translated into implementation-specific formats for distribution.

Figure 8.6 explains the XML Schema for interface definitions on the example of Ethernet. Currently, there the Schema defines the interface types Ethernet, WLAN and serial (modem). A `hint` section inside the interface allows to explicitly specify the real hardware interface corresponding to this definition. Otherwise, the interfaces are just assigned in the order they appear in the configuration file and on the computer system. Hints can be given for systems with PC architecture (numerated interfaces) or with Junos and Cisco appliances (interfaces sorted per pic, port and slot).

It is possible to configure one interface to listen to more than one IP address. This is supported by means of the `nw` elements inside the binding. The hardware section also defines the client side for the network helper protocol DHCP, while the server side of that protocols is configured as a service. The nw element either enables DHCP on the interface or has to provide a list of static settings like

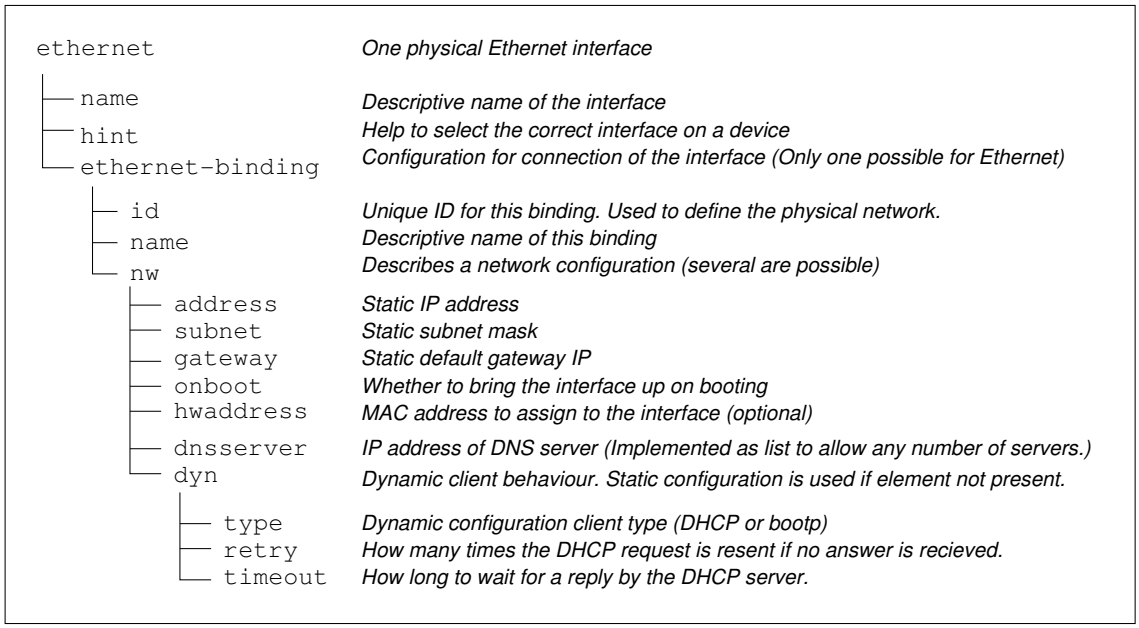


Figure 8.6: Configuration definition for network interfaces.

IP address. The static settings are configured in attributes. The `dnsserver` is an exception: it is an own element, because there may be several DNS servers for one interface. Some systems allow to specify an arbitrary number of DNS servers for one network interface. In real situations, there are typically no more than two or three DNS servers, which is fine with most implementations. Verinec allows an unlimited number of `dnsserver` elements. The MAC address of an interface can be set manually using the `hwaddress` field. The Ethernet standard recommends to only set local MAC addresses manually. The scope is defined by the 7th bit of the address. Universal is 0, 1 signifies local. Verinec does not force the administrator to only assign local addresses but allows universal addresses too.

WLAN differs from the Ethernet by additional attributes on the binding element. They define the SSID to use and possibly a pre-shared WEP key. By using different session keys, it is possible to "connect" one WLAN card to more than one network. A modem can also connect to different networks, dialling another telephone number. To reflect this situation, WLAN and serial interfaces are allowed to contain more than one binding. For the physical network layout, it is not the interfaces that are connected directly, but the *bindings* they contain. Therefore, the *bindings* are identified with an unique ID attribute. Ethernet cards can only be connected physically to one single network. Thus Ethernet interfaces have only one binding.

Advanced TCP / IP settings as discussed in Section 3.2 are not supported by Verinec because they are rarely used.

8.2.2 Network

The network definition of Verinec is rather simple. Physical networks are defined by specifying the bindings belonging to the same segment. In Ethernet, this would signify they are connected to the same hub. Collections of bindings referenced by their ID are grouped in `physical-network` elements under the container `physical`. Of course, only interfaces of the same type may be grouped. Each binding belongs to exactly one interface, which in turn is defined inside its node. This defines the network membership(s) for each node. The physical network consists only of interconnected interfaces (hub for Ethernet, common session for WLAN). Every device on a higher level than a hub, e.g. a switch, is considered a node with several interfaces, each belonging to one network.

To specify the logical structure of the network, physical networks can be grouped with the `logical-network` element. A logical network might be the subnet of one office or floor, or the

network of one organisational unit. Listing 14 shows a sample network. It consists of two groups of connected interfaces and a logical network spanning over the two groups.

Listing 14: A small network

```
<networks>
 <physical>
  <physical-network name="intr1">
    <connected binding="clientA-i1"/>
    <connected binding="clientB-i1"/>
    <connected binding="Server-i1"/>
  </physical-network>
  <physical-network id="extr1">
    <connected binding="Server-i2"/>
    <connected binding="Router-i1"/>
  </physical-network>
 </physical>
 <logical>
  <logical-network id="log" name="smallnet">
    <description>
     A small network consisting of an internal part
     and a DMZ for the Server.
    </description>
    <lphys idref="intr1"/>
    <lphys idref="extr1"/>
  </logical-network>
 </logical>
</networks>
```

To know what the network really looks like, the nodes have to be known as well. Only the nodes define which bindings belong to the same node. Assume there are two clients with interface id's clientA-i1 and clientB-i1 respectively, a server with interface id's Server-i1 and Server-i2 and a Internet router with interface id Router-i1. Then the above network description defines an Intranet connecting the clients to the server, which is in turn connected to the Internet router. Figure 8.7 shows the network in the graphical editor.

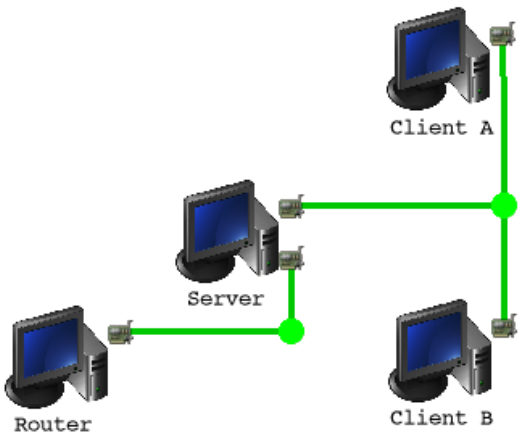


Figure 8.7: The small network visualised in the editor.

Currently there is no notion of network latency (cable length, signal retardation) or capacity because such features are not implemented in the simulator.

8.2.3 Variables

The configuration format presented in this section so far defines all necessary information to configure the network. In order to achieve error-free configuration, further improvements can be done. Redundancy in configuration data should be avoided as much as possible. If a network contains three similar routers, it is boring to change their configuration individually and if the administrator forgets one of them, configurations start to diverge. For a computer room with 100 equal workstations, it is practically impossible to administrate each one separately. There exist tools to clone installations, but for Verinec, a more sophisticated concept has to be found.

Each value should exist in only one place, otherwise there is a risk that a setting is updated in some places, but left unchanged in others. Examples of values appearing in several places are the IP for the DNS server in static interface configuration or the own IP of a host in firewall rules. This information can be copied for every occurrence in the node configuration. But when something changes, the administrator has to update every copy of the setting. This is not just a waste of time but also a severe threat to the reliability of a network. It is just too easy to forget to update some copies.

To provide for more flexibility, Verinec supports *variables* in the attribute values of node documents. The variables are useful to isolate common information into one place, allowing changes to be more painless. For example, in a network with many clients all using the same DNS server, the DNS server IP can be defined as a variable. If for some reason, the IP has to to be changed later on, it has to be updated in only one place.

Variable declarations are allowed everywhere in the nodes document and can be referenced from any attribute.[3] Variable references are resolved in lexical scope of the XML document, meaning an ancestor element has to contain the variable declaration. If a referenced variable is not defined in the context of its usage, document validation fails.

To reference a variable, the variable name is written in the attribute, enclosed in dollar signs, like `$variable$`. To use a literal $ sign in an attribute value, it has to be escaped in the XML like '\$'. The variable name may consist of upper- or lowercase letters, digits and the underscore.[4] A variable is defined using the `variable` element `<variable name="`*variable*`" value="`*value*`" />`.

There is one exception of variables not needing to be defined in the XML document. System variables are predefined outside the XML document by the Verinec application. The names of those variables begin with the percent sign %. They are resolved against the Java environment. For example, '$%user.name$' will result in the username of the current user.

Macro expressions are used to reference special values. A macro is specified as `{!macroname}`. Currently, there is only the `hostname` macro available. Section 10.2 proposes more sophisticated models to improve configuration definition further, including an extension to macros using XPath expressions.

8.3 Services

Besides having interfaces, each node can provide services for the network. The services are configured in the second part of the node definition. Each service is enclosed in a tag with its service name. To explore the Verinec concept, support for some network services has been implemented. The general idea of the services presented here have been introduced in Chapter 3. This section discusses their implementation in Verinec.

8.3.1 Routing Configuration in Verinec

The Verinec configuration for routing is only specified for static routing. To provide for extensions in the future, an element `dynamic-routes` is defined, but empty for now. To support protocols like RIP, OSPF or BGP, according XML elements for dynamic-routes would have to be defined.

Route filters similar to the packet filters (see next section) allow to define conditional routing based on packet properties. There are however less tests in routing than for filtering: only destination and source of the packet can be taken into account and routing metrics and the next hop

[3] Note that the XML Schema knows nothing about variables. The framework resolves all variables prior to configuration verification. Otherwise the Schema could no longer enforce strict formats for attributes.

[4] The regular expression is `(a-zA-Z0-9_)*`.

the packet would take. The actions can change the next hop or the metric and accept or drop the packet. The actual routes are expressed under the `static-routes` element. Figure 8.8 gives an overview of the XML Schema.

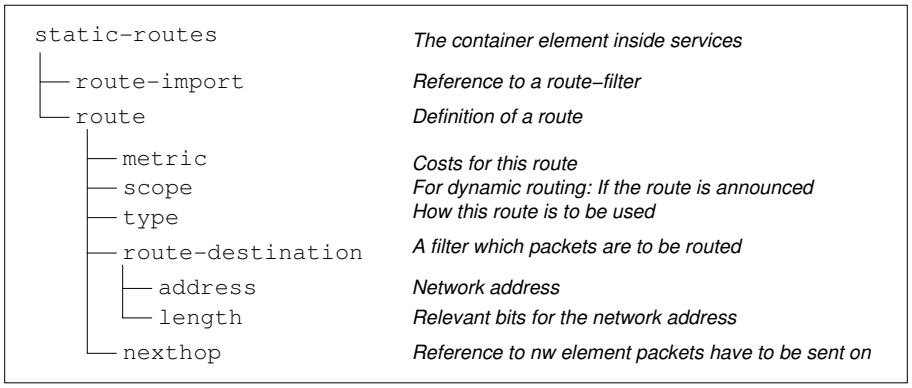


Figure 8.8: Elements of the routing definition.

Route filters are referenced using the route-import element. The actual routes are given a `metric` value for priority decisions. Their `scope` is relevant to decide if they are announced when dynamic routing is used.

link: The route is local to this link.

site: The route is advertised to neighbours, but should not leave the network.

global: This is a route external routers might be informed of.

The `type` decides what to do with the packets on that route. The following types exist:

normal: This is a normal route.

local: This route is only for the local network.

reject: The packet is discarded and the sender informed that the network is not reachable.

prohibit: The packet is discarded and the sender informed that the network is prohibited.

blackhole: Packets taking this route are silently dropped, the sender is not informed.

The `route-destination` element is used to limit the route to certain subnets. The `nexthop` element finally tells which output interface this route is using.

8.3.2 Packet-filters Configuration in Verinec

Verinec models packet filtering in a generic manner, but it is clearly inspired by the *Netfilter* framework [Welte et al., 2006]. The XML elements and attributes are listed in Figure 8.9. The `packet-filters` element is used as container within the services list. It has attributes for references to the three chains for input, output and forward, with the same semantics as in the Netfilter framework. The children of packet-filters are `packet-filter-chain` elements, each defining a chain of rules. A chain is always referenced by its id, which thus has to be unique, while the name is only used for better readability of the configuration. Each chain has a default policy for packets that did not match any rule.

The actual rules are defined in the `packet-filter-rule` elements. Each rule contains a `packet-match-list` with criteria and an `packet-action-list` with the action to take in case all criteria are satisfied. The match-list is a subset of the tests possible with Netfilter/iptables. All tests have a 'negate' attribute to match "anything which does not match the criteria". The available match criteria are:

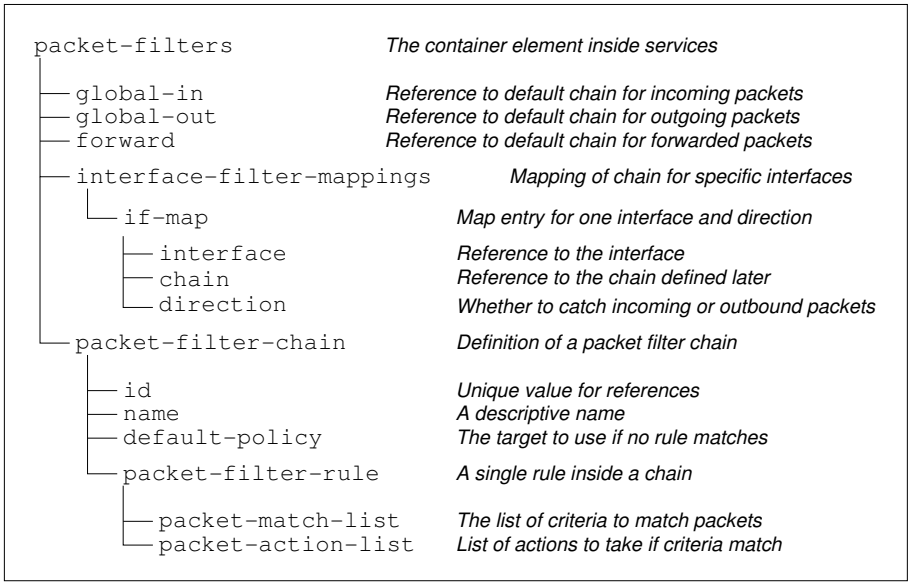


Figure 8.9: Elements of the packet filter definition.

match-in-interface: The interface the packet was received from (not used in OUTPUT table).

match-out-interface: The interface the packet is going to be sent from (not used in the INPUT table).

match-mac: Match an Ethernet MAC address.

match-ipv4: Test on IP criteria. Child elements for source and target IP address as well as for other header values.

match-tcp: Test on TCP criteria. Child elements for source port and destination port range as well as flags and options.

match-udp: Test on UDP criteria. Child elements for source port and destination port range.

match-icmp: Test packets of the ICMP protocol. Child elements to specify ICMP type and and code.

match-state: For stateful inspection. Can match packets creating *new* connections, packets belonging to *established* connections, packets *related* to an existing connection and packets not attributed to any connection, called *invalid*.

The test criteria are hierarchic. Tests only applicable for IP traffic are inside the base `ipv4` element. Source or target port tests are only usable inside tcp or udp criteria. The XML format allows to enforce proper parameters in a straightforward way using XML Schema.

If all tests in the 'match-list' succeed, the action of the rule is executed. The 'action-list' contains exactly one of the following actions:

noop-action: Do nothing and continue processing. Useful when only the log-action is wanted.

accept-action: Accept the packet and stop processing.

drop-action: Silently drop the packet.

reject-action: Drop the packet and send an error message to the sender.

gosub-action: Jump into a user defined chain.

return: Return from this chain. If this is a built-in chain, the default action is executed.

Optionally, the action-list may contain an additional `log-action` along with one of the above actions. This differs from iptables, where logging is an action of its own right. For iptables, taking an action and logging the same event can only be achieved by duplicating the rule.

Another feature of Verinec XML not coming from the Netfilter framework is attributing chains to interfaces in the `interface-filter-mappings` list. As illustrated in Figure 8.9, this is a list of explicit mappings between chain identifier and interfaces id. Although Netfilter does not know such an explicit mapping, explicit specification of interface chains is preferable. It allows for cleaner translation of the configuration for implementations supporting interface specific chains. Cisco for example only knows per-interface chains and no global chains. In Netfilter and other systems without explicit mapping, this feature has to be emulated. In the FORWARD chain and INPUT or OUTPUT chain - depending on the `direction` attribute - tests for that interface are added at the beginning. On a match, the rule jumps to the target chain referenced in the map. If an interface chain defines no default policy and just returns in the end, the global chain will come to action.

8.3.3 DNS Server

The Domain Name System (DNS) configuration format in Verinec is derived from the BIND implementation. The mapping is thus straightforward for BIND [ISC, 2007] importers and translators. Examples of BIND configuration files have been shown in Section 7.2.4. The root for DNS is the `dns` element. It may contain any number of `Zone` children. Figure 8.10 shows the XML Schema for a zone in Verinec.

`Zone`	*Container of one zone the dns server is responsible for*
├── `match`	*The zone name*
├── `primaryns`	*Host name of the primary DNS server for the zone*
├── `adminmail`	*Email of administrator in domain name encoding*
├── `serial`	*Version number of the configuration*
├── `refresh`	*Tells other DNS servers after how many seconds the zone has to be updated*
├── `retry`	*Frequency for refresh retries in seconds*
├── `expire`	*After how many seconds the zone data is to be considered expired*
├── `min_ttl`	*Default value for the Time To Live of the records*
├── `dnsNS`	*Entry of a name server for the zone*
│ ├── `match`	*Domain name to handle*
│ └── `target`	*Host name of the DNS server*
├── `dnsIPRange`	*A set of hosts in a zone*
│ ├── `network`	*Defines the network part of IPs in the group*
│ ├── `description`	*Human readable description*
│ └── `dnsA`	*A name to address line*
│ ├── `match`	*Either absolute or relative hostname. If not ending with dot, domain is appended*
│ └── `target`	*Local part of the IP address, to append to the part defined in network attribute*
├── `dnsTXT`	*Defines a text string to associate with a host*
├── `dnsCNAME`	*Defines a canonical name pointer to forward to an other host*
└── `dnsMX`	*Defines a mail exchange host for the domain*
├── `match`	*Domain name to handle*
├── `target`	*Host name of the mail server*
└── `priority`	*Priority of this mail server*

Figure 8.10: Elements of the DNS server configuration.

The attributes directly at the root correspond to the header section of the zone declaration. The elements beginning with `dns` result in lines for the body of the zone declaration. `dnsIPRange` does not come from BIND. It is used to group hosts in the same subnet and encapsulate information common to all entries of that subnet. Usually, all hosts in a zone belong to the same network. But in a more complex setting, this might not be the case. Verinec allows to have several ranges in one zone. The actual mapping between host name and IP ('A' records in DNS terminology) is done in

the `dnsA` records.

The other entries exist on a per zone level and are not further grouped. Textual annotations about the domain can be set in `dnsTXT` records. The `dnsCNAME` record allows to forward one host name to another one. Finally, `dnsMX` is used to specify the servers accepting emails for addresses at the domain. Using different values in the `priority` attribute, mail transfer agents (MTA) are instructed which mail server to try first when delivering an Email for the domain.

Some DNS implementations, i.e. BIND, require explicit reverse DNS entries in their configuration files. In Figure 8.10 there are no entries for reverse DNS lookups. Verinec does not need them, as it would be redundant information. It can be deduced from the `dnsA` records, as it is just the inverse mapping of IP to host name. Translators for implementations expecting explicit reverse DNS entries can generate everything necessary from the A records. Generating the configuration files is discussed in Section 8.5.5.

8.3.4 DHCP Server

For Dynamic Host Configuration Protocol (DHCP) server configuration, a couple of configuration settings is required. At least, the server needs to know the range of IP addresses it is allowed to give to clients. Figure 8.11 specifies the main components of the DHCP configuration. The Verinec definition is inspired by the ISC DHCPd [ISC, 2007], which is mostly visible in the names of attributes and elements. However, most of the features are available in any DHCP server implementation.

`dhcp-options`	*Options applicable at many places*
├── `default-lease-time`	*Time to offer a lease if client does not specify a preferred length*
├── `max-lease-time`	*Maximum lease time to accpet if client specifies a preferred length*
├── `authoritative`	*Whether hosts in this scope should receive authoritative responses*
├── `dhcp-dns-list`	*List of DNS servers for a client in this scope*
└── `dhcp-router-list`	*List of routers for a client in this scope*
`dhcp`	*The container element*
├── `shared-network`	*Allows to run more than one network on one physical interface*
│ └── `name`	*Purely descriptive name*
├── `group`	*Group containers to specify common parameters*
├── `subnet`	*A subnet (Is a container)*
│ ├── `domain`	*Domain name for this subnet*
│ ├── `addr`	*Subnet address*
│ ├── `netmask`	*Subnet netmask*
│ └── `range`	*A range of IP addresses that can be used*
│ ├── `begin`	*First IP*
│ └── `finish`	*Last IP*
├── `pool`	*A pool of addresses*
│ ├── `pool-restriction`	*Specific restrictions for a pool*
│ └── `range`	*A range, as defined above*
└── `host`	*A explicit host statement to define mac addresses and fixed mappings*
├── `name`	*Name of the host*
├── `host-hardware`	*Specifies the host hardware to recognise a client*
└── `fixed-address`	*Specifies an IP address to assign to the host*

Figure 8.11: Elements of the DHCP server configuration.

The configuration is hierarchic, with the possibility to specify parameters on every level of detail. Containers can group other containers, which inherit their parameters. Subnet, group, pool and shared-network are containers. A container can group one or several groups, subnets, pools and hosts. A couple of general options can be configured for each container. The default

and maximum `lease times` control the time after which clients have to renew their configuration, possibly receiving a changed IP. The lease time is configured in the server, which tells the time to its clients in the DHCP offers. The `authoritative` attribute tells the server whether it should send `DHCPNAK` messages to clients asking for renewal of IPs from another network. If the DHCP server is properly configured for a network, it should be set authoritative. Authoritative is off per default, to prevent unconfigured DHCP servers from confusing the DHCP system with `DHCPNAK` messages. `dhcp-dns-list` and `dhcp-router-list` inform the server about the addresses of DNS servers and routers to tell the clients. This allows them to really use the network after automatic configuration. The DNS server should be informed by the DHCP server about the host names and IPs the later issues, to allow other clients to look up the hosts. This is configured with attribute ddns-updates in one of the containers or globally in the `dhcp` element. Further specification for the update process is given in the attributes ddns-hostname and ddns-domainname. Other settings can be specified using the `dhcp-restriction` element. It specifies whether to allow or deny clients matching a certain criterion. It applies to the container it is in and is inherited by child containers, unless the same criterion is overwritten there. Each criterion can either be set to *allow*, *deny* or *ignore*. Ignore is the same as deny, except that ignoring produces no entry in the log file. Possible criteria to control are:

unknown-clients: Can be used to place clients not having a matching `host` declaration onto a separate subnet. To achieve this, two subnets are declared. One is for the known clients (type="deny" criteria="unknown-clients") and one for the unknown clients (type="allow" criteria="unknown-clients").

bootp: Tells the server whether it should respond to bootp requests or not.

booting: Only relevant in the context of a host declaration. If it is denied, this client is not given an IP, allowing to disable single clients.

duplicates: Used to prevent one client from blocking several IP addresses when it uses different DHCP Client Identifiers on one MAC address. This typically happens with dual boot systems. If allowed, the server discards all existing leases for a MAC address when it receives another request from that address.

declines: Controls whether the server processes `DHCPDECLINE` messages from clients. This message tells the server that the offered address is not valid for some reason. The server does not use that address any more. A malicious client could exploit this behaviour to exhaust the address pool of the DHCP server by always sending declines.

client-updates: Tells the clients whether they are allowed to inform the DNS about their new address. It should be denied if the DHCP server takes care of informing the DNS system.

A shared network groups several subnets which are present on the same physical interface. It can only occur directly under the `dhcp` element. It does not provide additional options in addition to the parameters a container can set. The container `group` element has no direct implications on a network. It is used to set common parameters to some subnet, pool or host declarations. The `subnet` is used to specify a part of the network with common information. The addresses available should either be specified in ranges, or using pools to better control what is available. The subnet uses restrictions to select the clients belonging to it. In a `pool`, IP addresses that are reserved for a particular purpose are grouped. Access to pools is controlled using `pool-restriction`, which is different from the `dhcp-restriction` element. Pools can be restricted to unknown or known clients, to clients using bootp or to clients being authenticated or only those not authenticated. The `host` finally allows to identify clients, typically based on the MAC address they provide. This allows to specify the list of known hosts. If desired, fixed mapping of IP addresses and host names can be defined.

The configuration for a DHCP *client* is trivial. The interface settings just have to specify whether that interface should be configured using DHCP or not. Section 8.5.6 contains some notes on translating DHCP server configuration.

8.3.5 Generic Service

It is not realistic to hope that *all* existing services are supported by the Verinec configuration XML Schema. Verinec should be able to somehow handle services it does not know. An Nmap port scan (see 7.1.3), for example, might reveal services not known to Verinec.

To cope with unknown services, the element `generic-service` is used as a meta-service. Contrary to actual services, it may occur more than once in a node configuration. It provides attributes to define whether TCP or UDP are used on transport layer and on which port they listen. For the human user, additional information about the service can be provided. Listing 15 shows a couple of generic services. The first service, "domain", is DNS which happens to be supported by Verinec. The second one is "ssl" and this service is currently not supported. This Schema part is based upon the Nmap XML output format [Fyodor, 1997].

Listing 15: A sample node

```
...
<generic-service protocol="tcp" portid="53">
  <port-state state="open" />
  <port-service name="domain" method="probed" conf="10" />
</generic-service>
<generic-service protocol="tcp" portid="443">
  <port-state state="open" />
  <port-service name="ssl" product="OpenSSL" method="probed"
     conf="10" />
</generic-service>
...
```

Using `generic-service`, it is possible to store some information about services not supported by Verinec. The simulator should also implement a generic service, supporting TCP connection establishement and replies on UDP requests. This does not allow to fully test a communication with all side effects. But at least it becomes possible to check for basic connectivity to the according ports.

8.4 Adaptation Module

The adaptation module is concerned with distributing the configuration to the devices. It has to adapt the abstract configuration to all kind of device and implementation-specific factors, hence its name. To keep the remainder of Verinec free from device specifics, all logic to handle implementation-specific issues is concentrated in the adaptation module.

The implementation of the adaptation module follows the considerations from Chapter 6. It implements the use case from Section 4.4. The adaptation process is divided into translation, restriction and distribution, as illustrated in Figure 8.12. To define which implementation a node uses, a type system is introduced. It tells the module which translator XSL document from the repository is appropriate to transform the configuration into a new XML document containing the definition of system specific configuration and some meta data. The meta data is later used by the distributor to restart services or perform other operations before or after modifying the configuration. For each translator, there is a *restrictor* that creates warnings if features in the abstract configuration cannot be translated for the target implementation. Restrictors and translators are selected in exactly the same way, ensuring that the system produces the warnings specific to the implementation currently targeted. The distribution method however is independent of the translation. Thus, there are two things the type system has to define: the translation document to produce implementation-specific configuration and the distribution method to connect to the machine to transmit the configuration. Translation/restriction are defined in a *service* declaration, the distribution is specified in a *target* declaration.

Note that in the adaptation context, the child elements of the 'hardware' part (Ethernet, WLAN, serial) are treated the same way services are. For distributing configuration, is does not matter whether a hardware driver or server software is configured.

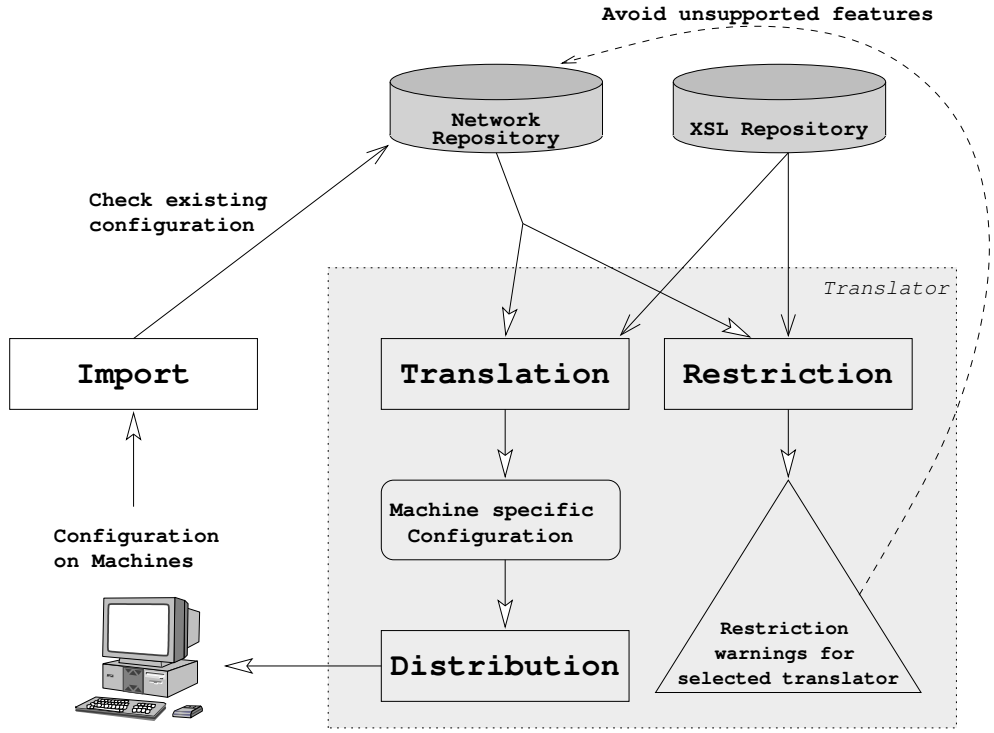


Figure 8.12: The distribution process.

8.4.1 Type System

The services of a node cannot be translated into implementation-specific instructions without a translator. Each node has to specify for every service which translation and target are to be used. The type system allows to group translations and targets for common implementations, for example for a Linux system running the Red Hat distribution. Before the actual translation, Verinec resolves the type information to determine which translator to use for each service of a node. The type system metadata in the abstract XML configuration is placed in a separate namespace and attached to the configuration. The namespace for the translation Schema is `http://diuf.unifr.ch/tns/projects/verinec/translation`. Its prefix is usually `tr`.

Listing 16 shows an example of a type declaration. Types are either defined globally in the `tr:typedef` element as `tr:type` or directly inside a node using the `tr:nodetype` element. The typedef element also specifies a default type to be used for nodes with no type reference. Nodes can reference globally defined types. Definitions for individual services can be overwritten as necessary. Type declarations contain `tr:service` elements to define which translator to use for what service. The service elements can in turn define a target to specify how the translated configuration will be distributed. The name of the target method is the name of the child element of `tr:target`, which is in this example `cp`.

Listing 16: Type declaration in the translation Namespace

```
<nodes ... xmlns:tr='http://diuf.../verinec/translation'>
  <tr:typedef def-type-id='xy'>
    <tr:type name='linux-redhat' id='typ01'>
      <tr:service name='ethernet' translation='linux-redhat'>
        <tr:target name='copy to mnt'>
          <tr:cp prefix='/mnt' />
        </tr:target>
      </tr:service>
       ...
```

Declared types can be referenced from nodes, as shown in Listing 17. The `type-id` attribute in the nodetype references the identifier of the type, specified in its `id` attribute. The example references the type defined in the above listing. It also shows how to overwrite some of the service translators. The linux-redhat translator for the service ethernet is replaced with a translator named linux-custom. The translator for the WLAN service, which was not defined in the type declaration, is set to linux-redhat.

Listing 17: A node using typ01 and overwriting some translators

```
<node hostname='machine666'
    xmlns:tr='http://diuf.unif.../verinec/translation'>
  <tr:nodetype type-id='typ01'>
    <tr:service name='ethernet' translation='linux-custom'/>
    <tr:service name='wlan' translation='linux-redhat'/>
  </tr:nodetype>
  ...
```

Each node has to be completely resolved before it can be translated using the XSL transformer. All translator and target information is copied locally into the node's `tr:service` tags. This way, the adaptation module has to care only once about the type resolution and precedence rules and the XSL translators can copy the targets to the output documents. The precedence for translation definitions is straightforward:

1. Locally defined translator for that service.
2. Translator defined in the type referenced by the node.
3. Translator in the default type if the node does not specify a type.

Targets are used to specify distribution method and parameters. They can be set in several places. Besides the aforementioned declaration inside of a `tr:service` element, they can be defined globally inside the `typedef` element. Globally defined targets can be referenced from `tr:service` elements, to avoid redundant copies of targets. To specify which target to use as default for a node, targets can be included under or referenced by a `tr:nodetype` element. The typedef element declares which global target to use as default when no target is specified in its `def-target-id` attribute.

The rule of precedence for the targets makes sure that local settings overwrite global ones. Targets inside a service element always overwrite the ones in node element and the global default. There is only one point that might be surprising: a target declaration inside a service element of a `tr:type` has precedence over the target specified as default for a nodetype. The reasoning is that global types will only specify the target specific to a service if that service needs special treating.

1. Service in nodetype target child element
2. Service in nodetype target-id attribute
3. Service in global type target child element
4. Service in global type target-id attribute
5. Nodetype def-target-id attribute
6. Global default target set in typedef

Listing 18 shows some of the possibilities. *Serial* has a locally defined target, which has precedence. Note that if the translator does not change, it needs not be specified again in the nodetype when overwriting the target. The `tr:service` element can contain only the attribute `name` and the child element `target` or `target-id` attribute. The service *ethernet* inherits the target 'copy to special' specified in the ethernet service declaration of type `typ02`. As of the order of precedence, this target is used and not the one specified by nodetype's `def-target-id`. Service *wlan* however

has no specific target, the def-target-id of the nodetype is used (which points to `tar02`, 'copy to pc02'). If the nodetype would not specify a default target, the default target of the `typedef` element would be used, resulting in `tar01`.

Listing 18: Where to specify Targets

```
<nodes xmlns:tr='http://diuf.unif.../verinec/translation'>
  <tr:typedef def-target-id='tar01'>
    <tr:type name='sample type' id='typ02'>
      <tr:service name='ethernet' translation='linux-redhat'>
        <tr:target name='copy to special'>
          <tr:cp prefix='/special'/>
        </tr:target>
      </tr:service>
      <tr:service name='serial' translation='wvdial'/>
    </tr:type>
    <tr:target name='copy to temp' id='tar01'>
      <tr:cp prefix='/tmp'/>
    </tr:target>
    <tr:target name=copy to pc02' id='tar02'>
      <tr:cp prefix='/mnt/pc02'/>
    </tr:target>
  </tr:typedef>

  <node hostname='pc02'>
    <tr:nodetype type-id='typ02' def-target-id='tar02'>
      <tr:service name='ethernet' translation='linux-fedora'/>
      <tr:service name='serial'>
        <tr:target name='serial'>
          <tr:cp prefix='/mnt/pc02/serial'/>
        </tr:target>
      </tr:service>
      <tr:service name='wlan' translation='linux-redhat'/>
    </tr:nodetype>
    ...
```

There are circumstances where nodes exist in the network that should not be managed by Verinec. The adaptation module will complain if a node has no type. A small trick helps to handle the situation. There exists a translator named `none` for each service. It outputs an empty element, which will be ignored by the distribution framework. Listing 19 shows a type having all services translated with none. Nodes that should not be configured by Verinec can simply be set to this type.

Listing 19: Where to specify Targets

```
<tr:type name="none" id="no-translation">
 <tr:service name="ethernet" translation="none"/>
 <tr:service name="serial" translation="none"/>
 <tr:service name="routing" translation="none"/>
 <tr:service name="packet-filters" translation="none"/>
 <tr:service name="dhcp" translation="none"/>
 <tr:service name="dns" translation="none"/>
 <tr:service name="generic-service" translation="none"/>
 <!-- ... all services -->
</tr:type>
```

8.4.2 Translation and Restriction

The translators are responsible for transforming the abstract configuration into a format compatible with the target system. They are implemented in eXtensible Stylesheet Language Transformations (XSLT), which is the method of choice to transform XML documents. The generated document has to conform with the `configuration` Schema. Listing 4 shows an example configuration output for a network interface in fedora Linux. A `configuration` is a collection of `service` elements[5], one for each service. A `service` contains the configuration data and the `tr:target` element copied from the translation meta data and may also specify pre- and post-processing commands.

Listing 19: Where to specify Targets

```
<configuration xmlns="http://diuf.unif.../verinec/configuration"
            xmlns:tr="http://diuf.unif.../verinec/translation">
 <service name="hardware">
  <tr:target name="pc07"><cp prefix="/mnt/pc07" /></tr:target>
  <post-process>
    <command execute="nohup /etc/init.d/network restart"/>
  </post-process>
  <result-file filename="/etc/sysconfig/network-skripts/ifcfg-eth0">
# Ethernet card configured by verinec: ethernet 0 [eth0]
DEVICE=eth0
IPADDR=192.168.0.55
BOOTPROTO=none
TYPE=Ethernet
ONBOOT=yes
</result-file>
 </service>
</configuration>
```

Possible formats for the configuration data are:

result-file: Write the unstructured content of the element as file under the specified location in the file system. The path is specified as absolute path, but targets can add a in more complex scenarios.

result-wmi: Container for WMI commands. The commands are specified in XML and have to be interpreted by a WMI distributor. Works only with the wmi target.

result-xml: Allows to include arbitrary XML fragments into a configuration. A distributor implementation understanding this XML format has to be chosen accordingly.

The result file specifies the path on the target machine. If a distribution supports the same format, but expects the file to be at a different location, the translator does not need to be copied completely. A translator can be written to use `<xsl:include>` and just overwrite the pattern generating the result-file element, setting the correct path.

For each translator, there is a matching restrictor. The restrictor has to provide information about features its translator cannot handle, usually because of limitations in the software implementing the service. If the translator can cope with all features of the abstract configuration, the restrictor has to exist nonetheless, but just confirms that everything is fine. The restrictors are XSLT documents as well. They scan the abstract configuration for occurrences of unsupported constructs. The important fragment of the input document is shown in Listing 20. This action list tells the packet filter to reject the packet with an ICMP "host prohibited" message.

[5]Do not confuse them with the `tr:service` element of the translation namespace, used to declare translators.

Listing 20: Packet filter fragment

```
...
<vn:packet-action-list>
  <vn:reject-action type="3" code="10" />
</vn:packet-action-list>
...
```

Listing 21: A restriction warning

```
<restrictions xmlns="http://diuf.unif.../verinec/restriction">
  <service translator="packet-filters_cisco800S">
    <restriction srcpath="/node[1]/services[1]...reject-action[1]">
      Filter-chain pfc3: Not possible to specify ICMP type nor code
      on reject action with Cisco ACL.
    </restriction>
  </service>
</restrictions>
```

Listing 21 shows the output of the restrictor for the packet filter of Cisco IOS. The problem is that the packet-filters configuration defines the type and status code for the ICMP response of the reject action, while IOS does not permit to specify these details. The `srcpath` attribute of each restriction identifies the problematic configuration element unambiguously using XPath[6]. This can be used to locate the element and correct it. The current GUI implementation does not support editing service configuration, though.

Repository

The XSLT documents for transformer and restrictors are provided by a repository. The repository is created using the factory design pattern (see [Gamma et al., 1995]). This allows to use different implementations of repositories, i.e. using files or an XML database. The `TranslationRepFactory` is responsible of creating a repository. It looks at a system property[7] and tries to instantiate the class specified there. If the property is not set, it creates the default repository `XSLFileRepository`. This implementation of the repository reads the XSLT documents from files in the local file system. The directory layout is:

{basedir}/{service-name} ─┬─ /translators/*{name}*.xsl
                            └─ /restrictors/*{name}*.xsl

The directory *basedir* is resolved as follows. If an environment variable[8] is set, this is used. Otherwise, `./translation` is assumed, thus a directory relative to the current working directory of the application. *service-name* is specified by the Verinec configuration, i.e. `ethernet` or `packet-filter`. *name* is the name of the implementation as specified in the translation metadata. If there is no appropriately named XSL file to be found in the path, the implementation tries to load the resource `/res/translation/{service-name}/translators/{name}.xsl` from the Java classpath.

8.4.3 Distribution

Contrary to the translation, distribution cannot be implemented using only XML. For Verinec, each distribution target is implemented as a Java class. The interface `IDistributor` defines the required methods.

[6] XML Path Language [Harold, 2003] is a compact syntax for addressing portions of an XML document.
[7] `verinec.adaptation.repository.implementation`
[8] `verinec.adaptation.repository.xslfile_basedir`

A factory class is used to create the instances of distributors for the different targets. The factory implementation is specified by setting a system property[9] to the fully qualified classname of that implementation. According to interface `IDistributorFactory`, this factory must be able to create distributors implementing the `IDistributor` and initialise them. Verinec comes with the default factory implementation `PropDistFactory`. This implementation reads the file `/res/distributormap.properties` in the classpath to get a mapping of distribution target names to fully qualified class names implementing the distributors.

A distributor needs three methods. The `setTarget` method accepts a `<tr:target>` element with a child of the correct type. As the distributor is created specifically for that child's name, this will always match, unless the map is wrong. The `distribute` method is called to let the distributor process one result from the configuration. This can be as simple as writing a file in the local system or as complicated as performing a series of SNMP requests. The third method is `execute`. It is used for pre- and post-processing. The pre and post elements in the configuration document allow to specify commands to execute on the system before respectively after the distribution is performed. They are used for stopping and restarting services or otherwise inform the service implementation about the new configuration.

8.4.4 Extending the Adaptation Module

With the adaptation module being modular, support for new services can be added without writing Java code, as long as an existing distributor can be used. If a new distributor is needed for a system, a plugin framework allows to provide new implementation classes without modifying or recompiling Verinec.

Custom Translators

As mentioned before, the translation is performed using XSLT. A repository provides XSL documents for specified translator names. Along with the translator goes a restrictor, which reports all problems with configuration data that cannot be translated. Both are identified by service name and a name for the service implementation. Restrictor and translator for the same implementation have the same name. To write a custom translator, one needs to write an XSLT document to transform the abstract Verinec configuration into the implementation-specific configuration data.

The XSLT document has to be included according to the repository implementation. This is documented here for the default repository implementation, for other implementations, please refer to their documentation. As has been mentioned in Section 8.4.2, the XSLT File Repository maps the translator names to file names. The XSL is cached and never reloaded until application restart. Developpers should keep this in mind when bug fixing translators.

The general concept of a translator is to select its service configuration under `services` and apply templates to the children of its service. As mentioned above, one important action is to always copy the `tr:target` child of that service to the output document. There is a JUnit test named `verinec.adaptation.XslAllTranslationsTest` available which runs all existing translators and restrictors against a sample node configuration. This class can be useful to debug translators and restrictors.

Custom Distributors

For the most important remote configuration methods, distributors exist. When a new way of bringing configuration to a target device is needed, it can be added in form of a Java class implementing `IDistributor`. This interface has three methods:

- setTarget to initialise the distributor with target information as specified by the XML Schema.
- distribute for doing the actual distribution of a configuration element.
- execute to execute a command on the target system.

[9] `verinec.adaptation.distributorfactory`

The distributors should be developed as generic as possible. Service specific information should be generated by the translator. Distribution meta information is to be placed in the `tr:target` element. For example, the cp distributor does not contain any path, but gets it from the translator, which knows where its configuration file is usually stored. This way, `cp` for example can be used to copy any kind of file, be it network configuration, DNS or packet-filter rules.

To integrate a new distributor, the translation Schema needs to be extended with a new distribution method inside the target element. It has to specify all parameters needed to connect to the target system, like password, username and other implementation-specific information. Of course, all libraries used by the new distributor have to be added to the Verinec classpath.

For the default implementation `PropDistFactory` mentioned above in Section 8.4.3, the file `/res/distributormap.properties` is used to determine the implementation class of a target. To make the target usable, a mapping from target name to fully qualified class name of the new implementation has to be specified.

8.5 Existing Implementations of Translators

Translators have been implemented for some products. The description of these implementations groups them by service. Although the actual translation depends mostly on the service and the target implementation, all translators have a similar entry point. A template for the document root "/" selects the service this translator is for. The template for that service first creates the wrapping `<service>` element (of the configuration namespace, this time) and copies the target for that node type from the nodetype's service declaration. As mentioned previously, the type resolution copies the targets there for just this purpose. The example in Listing 22 prepares an Ethernet interfaces translator.

Listing 22: Typical initialisation of a translator.

```
<xsl:template match="/">
  <xsl:apply-templates select="vn:node/vn:hardware" />
</xsl:template>

<xsl:template match="vn:hardware">
  <service name="ethernet">
    <xsl:copy-of
select="../tr:nodetype/tr:service[@name='ethernet']/tr:target" />
    <xsl:apply-templates select="vn:ethernet"/>
  </service>
</xsl:template>
```

Red Hat Linux was the primary target operating system for the translation experiments. Red Hat supports all of the services implemented in Verinec. Note that Linux distributions derived from Red Hat are often also supported by the Red Hat translators, namely the free Red Hat variant Fedora. The same holds for Debian Linux translators, which can be used for Ubuntu and other Debian based distributions.

8.5.1 Generic Service

The `generic-service` element designates no real service. But there is no "magic" built into the translation framework. Generic service is treated like a normal service. This means the translation framework expects to find meta information to locate the correct translator and restrictor. As there is nothing to do, the `none` translator mentioned in Section 8.4.1 is used.

If generic services are used, which is typically the case after importing an existing network (see Section 8.8), each node type should declare the `none` implementation for generic-service. The restrictor will produce a warning that a generic service cannot be translated. Ideally, the Verinec user would replace all generic services with their actual services and provide configuration. If generic services are present on translation, it is good to warn before the distribution happens.

8.5.2 Hardware Interfaces

Interface configuration is highly distribution specific. But the number of features is manageable and translation straightforward. Verinec supports Red Hat, Debian, Windows XP and Cisco.

Red Hat Linux

Red Hat Linux uses file based configuration for its interfaces. As Red Hat expects one separate file for each interface, the translator generates several result-file elements, one for each ethernet-binding encountered. The files are located in `/etc/sysconfig/networking-scripts/` and named ifcfg-<interface-name>. In Chapter 7, the format of the files has been discussed in depth. For distribution, the cp and scp methods are suitable.

To notify the system of the new configuration, `/etc/init.d/network restart` has to be called in the post-process. As stopping the interfaces can interrupt the ssh connection, `nohup` is used. Nohup executes the program and does not stop if the user logs out before it is completed. Restarting the network interfaces produces some output on the command line, resulting in a warning written to the Verinec log file. As long as each line ends with "[Done]", the warning is irrelevant.

Debian Linux

Debian uses one single file for all interfaces. It is called `/etc/network/interfaces` and contains statements to configure all interfaces. Listing 23 shows a typical translator output for Debian. Post-processing is almost the same as for Red Hat, except that the init script is named networking instead of network. The command is thus `nohup /etc/init.d/networking restart`.

Listing 23: Network configuration for Debian.

```
<configuration>
  <service name="ethernet">
    <post-process>
      <command execute="nohup /etc/init.d/networking restart"/>
    </post-process>
    <result-file filename="/etc/network/interfaces">
#Network configuration linux-debian created by Verinec
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
     address 172.16.1.1
     netmask 255.255.0.0
     gateway 172.16.1.254
     hwaddr 00:0C:29:74:DC:60
     #peerdns not supported.
</result-file>
  </service>
</configuration>
```

Windows XP

Windows interface configuration is based on the registry. In Verinec, WMI is used for manipulating the network interfaces. Windows network configuration has been implemented by [Zurkinden, 2005]. The WMI distributor interprets XML specifications to script operations. Listing 24 shows the script to set the first network interface to a static network address. The `wmi` element specifies the WQL statement to retrieve the scripting object. Network interfaces are modified using the scripting object `Win32_NetworkAdapterConfiguration`. Methods on this object allow to set DHCP or static addresses, DNS servers and the gateway. Each `method` element indicates a method to be called with the parameters specified by `param`. The reason `param-value` is a

child element rather than an attribute of `param` is that the parameters expect arrays. There could be more than one param-value for the methods.

Listing 24: WMI script for network interfaces.

```
<configuration>
 <service name="ethernet">
  <result-wmi>
   <!-- Ethernet card configured by VeriNeC: Ethernet card eth0 -->
   <wmi class="Win32_NetworkAdapterConfiguration"
        namespace="root/cimv2" wqlKey="Index" wqlKeyValue="1">
    <method name="EnableStatic">
     <param name="IPAddress" type="ArrayOfString">
      <param-value value="172.16.1.1" />
     </param>
     <param name="SubnetMask" type="ArrayOfString">
      <param-value value="255.255.0.0" />
     </param>
    </method>
    <method name="SetGateways">
     <param name="DefaultIPGateway" type="ArrayOfString">
      <param-value value="172.16.1.254" />
     </param>
    </method>
   </wmi>
  </result-wmi>
 </service>
</configuration>
```

Cisco

Interfaces in Cisco routers have a special meaning. The first interface (Ethernet0) is the LAN side, while the WAN side is Ethernet1. Routers with several LAN ports incorporate them as an unmanaged switch on OSI layer 2 in front of Ethernet0. IP assignment and other network settings are only possible on OSI layer 3. Some devices use Ethernet2 in a three-legged firewall setup.[10] If the device supports port mirroring, that port can also have a higher name. Listing 25 shows the configuration commands for the LAN interface.

Listing 25: Cisco interface configuration.

```
ip default-gateway 172.16.1.254
Interface Ethernet0
  ip address 172.16.1.1 255.255.0.0
  ip access-group 101 in
  ip access-group 102 out
```

The translator written by [Ehret, 2005] generates configuration in Cisco IOS format and should be used together with the Cisco distribution method. For Cisco, the mapping of packet filter chains to interfaces is done at the interface declaration. Because there can be only one declaration block per interface, the ethernet translator has to generate the mapping of interfaces to chains. As Cisco uses numbers for the chains, the interface position is used to generate the chain number. Interface 0 has 101 for input filtering, 102 for output filtering, and so on. The actual filters are translated by the packet-filter translator.

[10] A three-legged firewall allows to get part of the security achieved with a DMZ: it contains a third port for the DMZ with its own filter rules.

8.5.3 Packet Filter

The packet filter is one of the more complex services configured by Verinec. Support has been implemented for the Netfilter framework omnipresent on Linux distribution and for Cisco routers.

Netfilter

The iptables application can be installed on all Linux systems. But as has been mentioned in Section 7.2.3, configuration formats vary. To solve this problem and avoid writing similar but slightly different translators because of that, Verinec does not directly generate the configuration file. It generates a `result-file` containing an executable script that configures iptables as desired.

Before the new configuration can be set, existing configuration has to be removed. The pre-process commands are `iptables -F` and `iptables -X`. The first deletes the rules in all chains, the second deletes all unreferenced chains. Afterwards, the target system's filter table is completely empty. Linux distribution specific methods are used to produce the final configuration. For Red Hat, there is an iptables script in the init.d directory. The result-file is copied to the target machine at `/tmp/iptables-init` and executed by the first post-process. Then the Red Hat `init.d` script of iptables is told to store the current configuration of iptables permanently by calling `/etc/init.d/iptables save`. The last post-process is `rm /tmp/iptables-init` to clean up. Debian has no iptables script in `init.d`. Iptables configuration has to be put into the folder `/etc/network/if-up.d/`, which is even simpler to do than the method for Red Hat. The target of the result file is the `if-up.d` directory and no post-process commands are required.

The translator has to take care of a couple of specific problems. If the target of a `gosub-action` (the -J parameter) is not yet defined, the script aborts with an error. Thus, all chains are created without content before the rules for each of them are defined. Because Netfilter does not support per interface chain, the translator has to add tests for interfaces in the interface-filter-mappings as described in Section 8.3.2.

To avoid unecessary jumps, the chain names INPUT, OUTPUT and FORWARD are treated as special cases. The straightforward translation would be to generate all chains and add jumps to the default chains to the chains referenced in global-in, global-out and forward attributes of the packet-filters. This would generate one unnecessary jump. Instead, the referenced chains are directly created as the respective default chains. Trying to create a chain with one of the build-in names produces an error anyway. If however, one of the default chains is also referenced from elsewhere, this could however lead to problems.

Cisco

Cisco's philosophy of assigning filter chains is rather different from Netfilter. There are no global chains, but each interface can be assigned one chain for incoming packets, one for outgoing. As has been mentioned, there are usually only two interfaces, one for the LAN, one for the WAN. Contrary to a Linux host, filtering packets addressed at or originating from localhost is not important on a router. The packets have to pass the 'in' chain of the interface they arrive on and the 'out' chain for the interface they are to be resent from.

The keyword to define packet filter chains in IOS is access-list. Until the most recent versions, the Cisco IOS could only use numbers and not names for access lists. In addition, the range of a number for an access list identifies its type. Lists with numbers between 1 and 99 are IP standard lists, from 100 to 199 IP access lists with extended functionality, 700 to 799 for MAC access lists and so on. Within Verinec, extended IP lists are used with a range from 100 to 199. As there are at most two access lists per interface, there is no risk of running out of available numbers.

The XSL translator was initially written by [Ehret, 2005], but had to be rewritten by the author of this thesis to operate correctly. Translation does not begin with the chains, but with the interfaces. For each interface, the interface-filter-mappings are checked to see whether an interface specific chain is defined. Of the global chains, only the forward chain is used, if it is defined. It is appended after both the in and the out chain for each interface. As the semantics of global-in and global-out chains are controlling traffic to the local host, they are completely ignored.

Chains referenced by the mapping or globally are duplicated for each reference. The reason is that Cisco access lists cannot check for the current interface. If the rule is only used by one interface, the test can be done at translation: if the chain is not for the interface in question, the

rule is left out - it would always evaluate to false. If the chain is for this interface, the rule is used without any interface test, as that test would always evaluate to true.[11] A more serious problem is that Cisco IOS does not support jumps from one access-list to another. The `gosub-action` can thus not be translated. Under one condition, translation can still be done. If the packet-match-list is empty, this is an unconditional jump and the chain can simply be copied in place. If there is a match-list, it is theoretically possible to copy the referenced chain in place, prepending each rule with the jump criteria. In practice, this is very difficult, as the Cisco syntax defines the order of the tests in a rule. And if there are cycles in the jumps, the translator would loop endlessly between the chains. The risk of endless loops in the translator is already introduced when expanding chains referenced without criteria, but at least such a cycle is probably a mistake, as it could provoke an endless loop when being executed by the filter implementation.

Rules for a Cisco device look like in Listing 25. As has been mentioned above in Section 8.5.2 on ethernet translation, attributing the chains to the interfaces has to be done in the ethernet translator. The translator here relies on the ethernet translation to produce the access-group statements. The configuration script is executed in the context of the existing configuration. To redefine the access-list, it has to be emptied first as explained in Section 6.2.3. Otherwise, the new tests would just be appended to the existing list. The translator only removes access-lists it is going to redefine. It is possible for the running-config to contain other access-lists, but as they are not referenced by the interfaces, they are not used and thus do no harm.

In this example, the Verinec configuration contains no interface specific chain for port 0, but a global forward chain. The forward chain contained only one rule, an unconditional jump to the RH-Firewall chain. The firewall chain is copied right into this chain. The rule after the comment `! match-state not supported` is an example for the risks of unsupported features. The rule `access-list 101 permit any any` originates from a test in the Verinec XML to accept all packets in states 'established' and 'related'. As this cannot be translated here, we end up with a rule accepting all packets, rendering the rest of the list unused. This is rather dangerous, as the default policy of the firewall chain would have been to reject (deny) all other packets. The last rule can never be reached, but this is not due to a bug. It is the default policy of the forward chain, which is overwritten by the policy of the firewall chain.

Listing 25: Snippet of a Cisco packet filter configuration.

```
...
! Ethernet port 0
Interface Ethernet0
ip address 63.236.73.147 255.255.255.0
ip access-group 101 in

no access-list 101
! Appending global chain FORWARD to in for interface eth0
! chain FORWARD
! chain RH-Firewall-1-INPUT
access-list 101 permit icmp any any 255
access-list 101 permit udp any any eq 5353
access-list 101 permit udp any any eq 631
access-list 101 permit tcp any any eq 631
! match-state not supported
access-list 101 permit any any
! match-state not supported
access-list 101 permit tcp any any eq 22
access-list 101 deny any any
access-list 101 permit any any
```

[11] Duplication of one chain used by different interfaces would not be necessary if it contains no interface tests at all. However, this would have been rather complicated to determine in XSLT and has not been done.

8.5.4 Routing

Routing configuration on non-router devices is usually not necessary. According to [Dooley, 2002], it is even bad design to have clients or servers participating in routing protocols. Usually, the gateway information found in static interface definition or provided by a DHCP response respectively is enough information for network connectivity. PC operating systems set their routes automatically based on the interfaces IP's and networks. Problems with the setup usually indicate a problem in the network rather than the requirement to change the PC routing information.

Translation has been implemented for Cisco routers by [Ehret, 2005]. The Verinec configuration does not define dynamic routing configuration, so only static routing is supported by the translator. For simplicity, `route-filters` have also been left out, although it would be possible to implement some of them using Cisco *routing policies*. The translator produces `ip route` commands, as shown in Listing 26. Each route matches on an IP and mask. The third IP number is the next hop, of which Cisco supports only one in static routing. At the end of route lines, a metric can specify the costs for that route. For dropping packets (blackhole routing), a special interface with the name `Null0` exists. It is used in the middle line of the example listing.

Listing 26: Snippet of a Cisco router configuration.

```
<configuration>
  <service name="routing">
    <result-file filename="cisco-routing">
ip route 10.10.10.0 255.255.255.0 63.29.19.19 10
ip route 192.168.0.0 255.255.0.0 Null0
ip route 0.0.0.0 0.0.0.0 129.32.23.10 12
    </result-file>
  </service>
</configuration>
```

Routing configuration has not been implemented for any PC operating system. On linux, static routing is done by calling the `route` program with the parameters for each route. This is usually done in a startup script. The location for the file is distribution specific, but the file contains calls to `route`. The route-filters have to be translated into iptables commands on linux. As the XML Schema fragment is inspired by the iptables, this would again be straightforward and very similar to packet filtering.

8.5.5 DNS Server

DNS configuration has been discussed in Section 8.3.3 Verinec contains translators for BIND and djbdns.

BIND

The configuration format of BIND (Berkeley Internet Name Domain) has been discussed in Section 7.2.4. The Verinec translator produces all necessary files for the configuration: the global `/etc/named.conf` as well as the files for each zone and reverse zones.

djbdns

The djbdns translator has been implemented by [Awesso, 2005]. djbdns [Bernstein, 2007] follows a somewhat different philosophy. Bernstein's software adheres strictly to the principle of "one tool one task". djbdns is actually a collection of tools. The for important programs are

dnscache: Treat client requests (forward queries to other DNS servers and cache responses for repeated queries).

tinydns: Provide a DNS server for a domain configured locally.

axfrdns: Handle zone transfer requests from other hosts.

Additional tools are used to compile the text-based configuration files into speed optimised database files. `dnscache` and `axfrdns` need no special configuration once they are installed. For `tinydns`, the translator generates the configuration file (usually simply called `data`) and invokes the program `tinydns-data` to build the binary database file `data.cdb`. All zones and global configuration is stored in one single data file. The output is shown in Listing 27. It is very compact, but just readable.

Listing 27: djbdns server configuration.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <service name="dns">
    <post-process>
      <command execute="tinydns-data" />
    </post-process>
    <result-file filename="/etc/tinydns/root/data">
#
# Configuration file for djbdns
#
.unifr.ch::siuftesun15.unifr.ch:86400
=siuftesun15.unifr.ch:143.21.201.100:86400
=sr-sax-81.unifr.ch.:143.21.214.81:86400
=diuflx74.unifr.ch.:143.21.57.74:86400
=mail.unifr.ch.:143.21.214.103:86400
=diufpc230.unifr.ch.:143.21.72.77:86400
@unifr.ch:mail.unifr.ch:86400
Cwww.unifr.ch.:sr-sax-81.unifr.ch.:86400
Cdiuf.unifr.ch.:diuflx74.unifr.ch.:86400
</result-file>
  </service>
</configuration>
```

Each line defines one property. The code characters are:

‘.’: A name server record and a start of authority.

’=’: Associates a name with an IP, including the reverse PTR. Usually called ‘A’ record.

’@’: A mail exchange (MX) record.

’C’: CName entries map host names to other hosts.

The ‘.’ would allow to also specify the IP of the nameserver. If the IP would be known in the `dnsNS` element, it could be included after the domain between the two ‘:’ for tinydns to create the A record automatically. But because of the Verinec configuration format, it is simpler to create a separate ‘=’ line (‘A’ record) for the name server host (here siuftesun15).

The post process calls the tinydns-data helper program to compile the database. There is no need restarting tinydns to notice the new configuration, as it monitors the configuration file for changes.

The configuration format is simple to parse for importing. Bernstein states the goal of a stable and easy format to be understood external tools explicitly in the documentation.[12] However, where BIND has a very complex format, the djbdns configuration is overly simplistic. The lack of structure would require the importer to guess on domains and IP ranges, as they are simply repeated.

[12] See `http://cr.yp.to/djbdns/tinydns-data.html`.

8.5.6 DHCP Server

There seems to be only one implementation of a DHCP server to be used on Linux systems, ISC DHCP. The translation has only been implemented for this product. The initial implementation had been written by [Awesso, 2005], but had to be thoroughly rewritten for Verinec. Section 8.3.4 discusses the configuration format used by ISC DHCP. As the format is based on the ISC implementation, translation is straightforward. After the configuration file is written, the server has to be restarted to load the new configuration. This is done in a post-process command. Listing 28 shows a DHCP configuration output.

Listing 28: ISC DHCPd configuration.

```
<service name="dhcp">
  <post-process>
    <command execute="/etc/init.d/dhcpd restart" />
  </post-process>
  <result-file filename="/etc/dhcpd.conf">
domain-name toons.foo.net;
authoritative;
option domain-name-servers ns1.unifr.ch ns2.unifr.ch;
subnet 10.254.239.0 netmask 255.255.255.224 {
  range 10.254.239.10 10.254.239.20;
  option routers rtr-239-0-1.unifr.ch rtr-239-0-2.unifr.ch;
}
subnet 10.5.5.0 netmask 255.255.255.224 {
  option domain-name internal.example.org;
  range 10.5.5.26 10.5.5.30;
  option domain-name-servers ns1.internal.example.org;
  option routers 10.5.5.1;
  option broadcast-address 10.5.5.31;
  default-lease-time 600;
  max-lease-time 7200;
}
</result-file></service>
```

Cisco routers can also act as DHCP servers. Translating DHCP for the Cisco distributor should not be very difficult, although more complex scenarios can probably not be mapped exactly.

8.6 Existing Implementations of Distributors

Currently, there exist targets for copying files, either to a local path on the machine Verinec is running on or to other machines using the *secure copy* method scp. For programs that are configured by user editable configuration files, scp is about all an administrator will need. For Microsoft Windows system, a distributor for using WMI is used. For Cisco devices, a special distributor using SNMP and TFTP has been implemented.

Distribution is not necessarily implemented in one single class. The `IDistributor` class is merely the entry point to the distribution method. Scp or WMI distribution bring a whole framework with them, which is used during distribution.

8.6.1 None target

In analogy to the `none` translator explained in Section 8.4.1, there is need for a distribution target that does nothing. The `none` target implementation just discards the information and does not fail even if there is no result at all.

In case of the `generic-service` presented in Section 8.3.5, or generally any service translated with the `none` translator, the `none` target is the only possible target. Every service needs a target

for the distribution system to work correctly. It can also be used to disable distribution of some service with a specific translator.

The `none` target as the `none` translator allow to handle such situations without adding code to the adaptation process to treat them as special cases. Such "magic" code for special situations, for example not treating code that has a translator name "none" would make the code less readable and could produce unexpected result if a user is not aware of the "magic".

8.6.2 Copy target

The `tr:cp` target uses the `DistCP` distributor implementation. This class expects one or several `result-file` elements. `result-file` has an attribute `filename`, which denotes an absolute file name, including the path on the target system. If `cp` specified a path in the `prefix` attribute, its value is prepended to the path. This is for example useful when the file system of the machine to configure is mapped into the file system of the management computer, or if all files should be written in a local folder for manual transfer. No further parameters are necessary for DistCP to know which file to writing the contents of the result-file element to. If the file is already existing, it is replaced with the new content.

Pre- and post-processing means simply using the value of the `execute` attribute as command to run on the local system. Note that external programs are OS dependant. But as it is the translator generating the pre- and post instructions, this is no problem.

8.6.3 Secure copy target

In most cases, the file system of a computer to configure is not mapped into the file system of the machine running Verinec. Configuration files need to be copied over the network. Encrypted communication is to be preferred. It protects login credentials and the file contents, which might contain further critical information. One method to encrypt data between computer systems is the Secure Socket Layer (SSL). On top of this transport layer, various protocols have been implemented. The protocol to replace insecure remote shells like rsh or telnet is ssh (secure shell). Besides a command line on the remote system, the ssh suite also provides the scp (secure copy) method to copy files from one system to another over an encrypted connection. Verinec uses the `DistSCP` class to perform secure copies. As `DistCP`, this class expects `result-file` elements, and writes the files to a remote system. The `tr:scp` target extends the `cp` target with configuration for the remote login. The Verinec implementation supports both public key and password authentication. Possible attributes for the scp target are:

username: A name to log into the remote system is needed both for public key and password authentication.

password: A password for the remote system. Either the password or the keyfile has to be set.

keyfile: Path to file containing the ssh keys for public key authentication. If omitted, a password is required. If the key file is protected by a passphrase, it must be given in the attribute passphrase.

passphrase: To access a protected private key in the keyfile.

host: Machine name of the target system.

port: Allows to change the port of the ssh server. Defaults to the standard ssh port 22.

hostVerification: During authentication, the remote host sends its public key. If the key is already in the list of known hosts, it is accepted automatically. Otherwise, there are different possibilities. If hostVerification is set to 'ignore', ssh completely ignores host identification, risking to disclose the password to an attacker machine imposing the original host. If set to 'interactive', the fingerprint of the key is displayed to the user who has to accept or reject it. The last possibility is to specify the fingerprint in the configuration.

remote-prefix: Path prefix on the target machine, in analogy to the cp target.

For Verinec, a pure Java implementation of ssh was required. The sshtools project[13] provides J2ssh, which does not only implement the ssh protocol but also provides an scp implementation. Other free implementations like JTA[14] hold no support for scp. Unfortunately, sshtools seems to be an abandoned project. The company behind the project is now selling a commercial rewrite of the library. A number of bugs can be encountered:

- Connection attempts with no target host name do not time out.
- After refusing a host fingerprint, the library hangs.
- The implementation is not capable to send the EOF character to remote system.
- The bug tracker on sourceforge[15] lists a large number of bugs that are marked unresolved.

There are several options to improve the situation in the future. If the administration machine uses a Unix-like operating system with the OpenSSH suite installed, the binary application scp could be executed. A more thorough solution would be to rewrite the `DistSCP` class to use more recent libraries like JSch[16] or "Ganymed SSH-2 for Java" [17]. Unfortunately, development of these projects started after the decision on which library to use in Verinec. Finally, distribution could be outsourced to the ant utility, as discussed in Section 10.2.

8.6.4 Cisco

Devices made by Cisco Systems Inc support the Simple Network Management Protocol (SNMP). As has been mentioned in Section 6.2.3, a first idea was to write a generic SNMP distributor to support all kinds of devices that provide an SNMP agent. However, in his master thesis, [Ehret, 2005] found out that the SNMP support of Cisco Series 800 devices is mostly limited to read access. SNMP is well suited for monitoring the device and gather statistics, but changing the configuration cannot be done with pure SNMP. Ehret ended up creating a Cisco specific distribution method using configuration file transfers.

The `DistCisco` class operates on `result-file` elements, which have to be formatted according to the Cisco IOS syntax. Configuration transfer will only be successful if the Cisco router is able to initiate a connection back to the Verinec station. The router does not implement the server part of the protocols, but can only act as client. DistCisco uses SNMP and the CISCO-CONFIG-COPY MIB to initialise a file transfer from a server running at the Verinec machine. For simplicity, [Ehret, 2005] chose to use the Trivial FTP (tftp) variant. This simple standard fits well for the task. [Ehret, 2005] implemented the tftp server in pure Java. Because of the limited capabilities of the protocol, the server implementation is very compact.

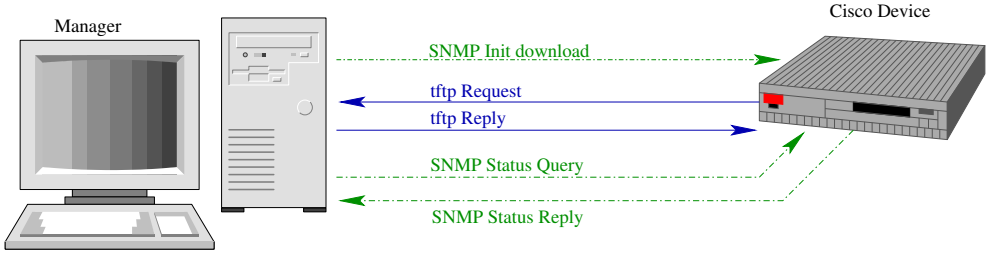


Figure 8.13: Cisco distribution implementation.

Figure 8.13 illustrates the process of a management station using SNMP to tell the agent on a Cisco router to download a new configuration file. Before distribution, the tftp server is started on the manager station. Then an SNMP command to trigger the download is sent to the agent on

[13] `sourceforge.net/projects/sshtools/`
[14] Telnet/SSH for the Java platform, `www.javassh.org`
[15] `sourceforge.net/tracker/?group_id=60894&atid=495562`
[16] See `www.jcraft.com/jsch/`.
[17] See `www.ganymed.ethz.ch/ssh2/`, by Christian Plattner and contributors.

the Cisco device. The full path to the configuration file, based on the root directory of the tftp server, needs to be told, as tftp provides no directory listings. This is no problem as the tftp server is under control of Verinec and the Cisco device can be told exactly where to retrieve the file from. The distributor polls on the state of the agent to notice when the transfer has completed.

Although SNMP plays a role, a separate `tr:cisco` target is used for `DistCisco`. This keeps the `snmp` target free for standard SNMP usage. The configuration payload in `result-file` is different from SNMP commands, directly containing statements in the IOS configuration language. The cisco target however *contains* a `tr:snmp` target specifying the connection parameters.

There is one drawback with the tftp distribution method and one problem with the CISCO-CONFIG-COPY MIB. With tftp, the configuration file is sent as clear text over the network (independent of whether encryption is used in SNMPv3 to trigger the transfer). It would be better to use scp or sftp instead. However, not all management stations provide an ssh server, and this is more complex than the tftp server to implement in Java. The author did not find any open source ssh *server* implemented in Java and did not want to implement one himself. This means that for using ssh, either the administrator would have to install an ssh server on the management station and make Verinec to correctly interact with it. This could cause problems, especially on Windows.

The second problem comes from the limitations of the Cisco MIB for triggering the tftp transfer. It is not possible to tell the Cisco agent to which port it has to connect to retrieve the configuration. tftp is running on port 69 per default, and Cisco devices simply rely on the server running on the default port. Unprivileged users do not have the right to start servers on ports below 1024. Thus either the tftp server has to be started by a privileged user outside of Verinec, which is complicated. Or the whole Verinec application has to be run by a privileged user, which is not very elegant. The last - and best - solution on Linux is to run tftp on a high port and redirect port 69 to that port using iptables. Then, administrator rights are only needed to set up the iptables rule. The NAT module is invoked with rule `iptables -t nat -A PREROUTING -p tcp --dport 69 -j REDIRECT --to-port <port>`

Verinec can be told on which port to start its built in tftp server by setting the environment property `adaptation.distribution.tftp.port`. If the port is 0, Verinec does not start the internal server. It expects an external tftp server to be running, and the system to be accepting connection requests on port 69. It is possible to start the Verinec tftp server as a standalone application, which would be used to run only the tftp server with administrator privileges. The command line is
`java verinec.adaptation.snmp.tftpserver.TFTPServer <port> <server directory>`. Alternatively, any other tftp server can be used. If the port property is not set, Verinec tries to start the internal server on the default tftp port 69. Otherwise, the server is started on the port number specified by the property. It is also possible to configure the root directory for the tftp server, the property is `adaptation.distribution.tftp.localpath`. If the port is 0 (external tftp server), the path property must be set, as Verinec cannot determine the root directory of the external application itself. Otherwise, `java.io.tmpdir` is used as default if the directory is not set.

As mentioned in Section 6.2.2, several versions of SNMP exist. The SNMP target supports both the community-based versions 1 and 2c as well as the more secure version 3. To improve readability, the target contains an attribute to specify the target host IP and children for either a community string or SNMP version 3. Version 3 needs attributes to specify username, passwords and the encryption method to use. SNMP communication has been implemented using the SNMP4J library [Fock and Katz, 2006]. This library offers complete support for all versions of SNMP and is well documented and maintained. The Verinec implementation only uses 1 and 2c for now, because it is less complex.

The Cisco distributor cannot execute arbitrary commands on the Cisco device. It however has one command implemented that is interpreted by the distributor: `cisco-delete-all-acl`. When it is specified in a configuration document, the distributor tells the Cisco device to upload its running configuration to the Verinec tftp server. Once the transfer is finished, the distributor parses the file for access-list declarations and generates `no access-list` statements for each list number it found. But, as has been stated above, this operation is not necessary when overwriting an existing configuration, as only unreferenced and thus unused access-lists will be left undeleted.

8.6.5 Windows Management Instrumentation

To automate windows configuration tasks, Microsoft provides Windows Management Instrumentation (WMI). This standard has been introduced in Section 6.2.4. For Verinec, WMI is used, although there is no pure Java implementation of WMI the authors know of. Distribution to Windows devices can only be performed if Verinec runs itself on a Windows machine. `DistWMI` uses the Jawin library [Halloway and Gehtland, 2005] to execute WMI commands. Jawin is a general integration layer for accessing Windows libraries through the Component Object Model (COM). It allows to use any scriptable component of Windows without the need to write a wrapper using the Java Native Interface (JNI). It comprises the Jawin Type Browser to generate stub classes for a library. This has been used by [Zurkinden, 2005] to generate all necessary classes to write WMI scripts in Java. The distributor simply retrieves the classes indicated in the `wmi` XML elements and calls the specified methods on them.

As its name suggests, Jawin can only be used on Windows. This means that the Verinec application needs to run under Windows, at least in a virtual machine, in order to use the WMI distribution method. According to [Zurkinden, 2005], there exists no multi-platform WMI client. There are WBEM frameworks implemented in pure Java, but they use HTTP as transportation protocol. For reasons mentioned in Section 6.2.4, WMI uses DCOM and provides no HTTP adapter. The only other alternatives would have been to implement some sort of Verinec agent to install on the Windows machines to configure. It could have been implemented in Java, using a library to access the registry[18] of that machine, or in .NET, using WMI only on the target machine. However, agents to install on target devices are against the design principles of Verinec, thus the Jawin solution was chosen.

The `DistWMI` distributor can distribute `result-wmi` files. Inside, result-wmi contains WMI function calls expressed in XML. This Schema part was originally developed by [Zurkinden, 2005] and has been integrated into the translation Schema. One or several `wmi` elements specify on which classes to operate and the WQL queries to select the desired instance. Child elements are used to specify the methods to call and their parameters.

The `tr:wmi` target (not to be confused with the resulting configuration `wmi` element) allows to specify how to connect to the machine to configure. The `host` attribute specifies the machine to connect to. `username` allows to override the user name to use when connecting to a remote system. Otherwise the name of the currently logged in user is taken.[19] The Windows domain can be indicated using the syntax user@domain. If there is a password required for the user to connect, it can be specified in the `password` attribute.

8.7 Verification Module

Verification requires information in addition to the network definition. To tell whether a configuration is semantically correct or not, Verinec needs a definition of what the semantics are. Test cases are used to specify requirements. This section presents the network simulator used to execute the tests and explains the implementation of network test cases. The section is concluded by revisiting the role model and explaining how the tests are used to define the roles. An elaborate discussion on verification implementation is found in [Jungo, 2008], who also wrote almost all code to implement the module.

8.7.1 The Verinec Network Simulator

The most important piece to test the network configuration is a simulator. A custom discrete time simulator has been built for Verinec [Jungo et al., 2004]. Based on events, it models the Internet layer model [Tanenbaum, 2003]. Every operation from transport to application layer is simulated and logged into an event tree. The simulation is started with a list of input events in XML format. Processing of those events can generate follow-up events which are scheduled at a later time. The XML event log is more accurate than is possible to get from a real network, as the

[18] For example the free implementations jRegistry Key `www.bayequities.com/tech/Products/jreg_key.shtml` or JNIRegistry `www.trustice.com/java/jnireg/`.

[19] For unknown reasons, the WMI framework does not support connecting to the localhost as a different than the current user.

causal chain is maintained. In log file analysis, it is difficult to decide for sure if an entry is caused by a certain event, particularly when events occur on different devices. In the Verinec simulation, this relationship is known and preserved in the event log by the element hierarchy. Events are added as child of the event that caused them. The output of the simulation can also be visualised in the GUI to get an idea of what is going on in the network.

An example simulation input is shown in Listing 29. At time 1, a UDP packet has to be sent on `client`. The `dst` attribute tells to ping to contact the node `server2`. On the destination machine, a "ping server" must be started explicitely. It responds to receiving an UDP packet by sending a UDP packet back to the sender. This allows to see whether communication is possible in both directions, which is usually desired. Although the program is called "ping" in Listing 29, it is actually an attempt to send a UDP packet to the host. Contrary to ICMP "echo request" messages, which are treated by the TCP/IP stack, UDP packets are only handled if there is some service running on the target. Note that the best solution would be to not start this service with an simulation input event, but automatically when a `generic-service` is present in the node definition. This should be implemented in a future version of Verinec.

Listing 29: Simulation Input

```
<events xmlns="http://diuf.unifr.ch/.../events">
  <header>
    <repository name="usecase" />
    <simulation stoptime="1000" />
  </header>
  <event time="0" node="server2" layer="5" service="application">
    <application program="PingUDPServer" type="service"/>
  </event>
  <event time="1" node="client" layer="5" service="application"
      dst="172.16.1.1">
    <application program="PingUDP" parameters="-b" type="launch"/>
  </event>
</events>
```

Listing 30 shows a fragment of the resulting output event tree. The sequence shows the client node sending an arp broadcast to determine the MAC address for the IP address of the server. After the arp reply has been received, the actual UDP packet will be sent to the server IP. The simulation log contains events from all layers. Even for the simple ping example, about 50 events where logged. Each transmission over the network is received by all nodes connected to the same hub. The simulation output is not very readable in its XML form. Verinec can however visualise the events in the GUI, helping to see what happens. Tests can be applied to the event tree to see whether the expected events occur, as discussed later in this section.

The simulator is built on the simulation framework DesomJ [Page et al., 2000]. This pure Java framework allows to combine the event-based with the process-oriented simulation approach. In Verinec, stateless protocols are implemented as event-driven modules, while stateful protocols are process-oriented. On top of the IP stack with TCP and UDP, the simulator implements DNS and generic TCP and UDP services to test handshakes. Below IP, ARP, Ethernet and "physical" data transfer are implemented. [Hug, 2006] implemented a complete packet filter for the simulator. As the other services, it directly uses the Verinec XML language as configuration. The simulator can be used to implement the use case from Section 4.1.

Listing 30: Simulation Output

```
<event time="1" node="client" layer="5" service="application">
 <application type="start" program="PingUDP" />
  <event time="1" node="client" layer="4" service="udp">
   <udp type="bind" />
  </event>
  <event time="1" node="client" layer="4" service="udp"
    src="172.16.1.100" dst="172.16.1.1">
   <udp type="send" />
  </event>
  <event time="7" node="client" layer="2" service="ethernet"
    src="00:0b:cd:47:a1:e4" dst="ff:ff:ff:ff:ff:ff"
    packetid="5e23bf9a:1143a41fe78:-7fcb" interface="eth1">
   <ethernet type="framesend" payloadtype="ADDRESS_RESOLUTION" />
   <reason xpath="/vn:nodes/vn:node[3]/...ethernet/@hwaddress" />
   <!-- nothing in physical layer for now -->
   <event time="9" node="server2" layer="2" service="ethernet"
     src="00:0b:cd:47:a1:e4" dst="ff:ff:ff:ff:ff:ff"
     packetid="5e23bf9a:1143a41fe78:-7fcb" interface="eth0">
   <reason xpath="/vn:nodes/vn:node[10]/...ethernet/@hwaddress" />
   <ethernet type="framereceived" payloadtype="ADDRESS_RESOLUTION"/>
   <event time="9" node="server2" layer="3" service="arp"
     src="00:0c:29:74:dc:03" dst="00:0b:cd:47:a1:e4">
    <arp type="REPLY" sha="00:0b:cd:47:a1:e4" spa="172.16.1.100"
      tha="00:0c:29:74:dc:03" tpa="172.16.1.1" />
```

Integrate the Simulator with Network Applications

To avoid having to write all servers and clients anew, the java.net infrastructure is used. Java creates sockets according to the Strategy design pattern [Gamma et al., 1995], enabling a flexible way to provide own implementations of sockets. The `java.net.Socket` and `ServerSocket` classes are wrappers around an instance of type `java.net.SocketImpl`. Using the static method `setSocketImplFactory()` on Socket and ServerSocket, it is possible to change the default implementation of the engine. [Jungo, 2008] implemented a factory to create sockets which do not send packets to the actual network, but into the simulation framework.

DNS queries are a special case. The resolution of domain names into IP addresses happens in the `java.net.InetAddress` class. It does not use the `Socket` class. In the standard Java framework, it is not possible to change the way name resolution is performed. However, since version 1.4, the Sun Java virtual machine allows to specify alternative name service implementations. The method is specific to the Sun implementation and is not guaranteed to stay in later versions.[20] The DNS configuration feature is not well documented in the official Java documentation[21], but a rather complete overview is provided by [Kuzmik, 2006]. First, the sun Java machine has to be informed about the class name(s) of available name service descriptors. This is accomplished by placing a configuration file with the name `sun.net.spi.nameservice.NameServiceDescriptor` into the folder `META-INF/services/` in the classpath. It simply lists the fully qualified class names of all implementations.

The specified classes have to implement the interface `NameServiceDescriptor`. They are factories to create `NameService` instances. Listing 31 presents a minimal factory. It is selected by setting the system property `sun.net.spi.nameservice.provider.X` to a value `"<type>,<name>"`, for example `"dns,mine"`. X is a number starting from 1, allowing to specify more than one provider. Sun's JVM cycles through the classes listed in the configuration file and uses the two get-methods to determine whether type and name match this property.

[20] Up to the current Java 1.6, it is still valid.

[21] `java.sun.com/javase/6/docs/technotes/guides/net/properties.html`

Listing 31: MyNSDescriptor.java

```
package fully.qualified;
public class MyNSDescriptor implements NameServiceDescriptor {
    public NameService createNameService() throws Exception {
        return new MyNameService();
    }
    public String getProviderName() {
        return "mine";
    }
    public String getType() {
        return "dns";
    }
}
```

`MyNameService` implements the interface `NameService`, which requires two methods to translate a name into a list of associated IPs or reverse translating an IP into a host name. The Verinec NameService implementation finally resolves names through DNS packets sent over the simulated network. On the other end of the simulated request, a node running a Java DNS server answers according to the XML network definition for that server.

Limitations of the Simulator

The Verinec simulator does not model network capacity and latencies. It would be interesting to consider the physical behaviour of network connections. This would permit to do performance analysis or test load balancing mechanisms. As discussed in Section 2.3.1, the bare existence of a connection does not guarantee reliable communication. If the network is congested with too much traffic, it is unusable even if connectivity would be possible.

Modelling latency would allow to create a new kind of tests that control whether requirements on the connection quality can be guaranteed. However, to model signal retardation and capacity, a database for the behaviour of typical network components is required. Even then it remains uncertain whether the predictions are correct, as latency depends on subtle factors. This area is probably better covered by tests on the real network.

Another feature not found in the Verinec simulator is simulating transmission errors. The impact of loosing some packets is mostly irrelevant for Verinec. This feature, found in classical network simulators like [ns, 2005], is useful to test the correctness of protocols. In Verinec, the protocols are assumed to be correctly implemented. But at some point, high collision rates affect the capacity and latency perceived at the end points because of the required retransmissions. If the simulator is extended to model capacity, collisions should also be taken into account. This could be achieved by modelling the capacity with Markov chains. This approach is taken in ns2 to avoid the almost impossible "exact" simulation of collisions and service degradation.

A master thesis by Philippe Der Hovsepian is currently exploring another approach to capacity testing. With the Verinec configuration being expressed in XML, it is not too difficult to translate it into input for the ns2 application. The simulation result from ns2 is then examined and the results presented in Verinec.

8.7.2 Constraint Language in XML

XML Schemas are well suited to specify the syntax of XML documents. However, there is no possibility to base constraints on actual element or attribute values. For example, it is not possible to formulate a constraint that *either* attribute A *or* child element B must be present, but not both. There exist several other XML description languages to do this.

Most famous among them is probably Schematron[22]. It was standardised by the ISO in the Document Schema Definition Languages framework [JTC1/SC34, 2006]. Schematron makes extensive use of XPath[23] . One XPath expression is used to *find* a node in the document. A list

[22] `www.schematron.com/`

[23] XML Path Language [Harold, 2003] is a compact syntax for addressing portions of an XML document.

of XPath expressions are evaluated relative to that node to *check* whether they are true. While Schematron is easy to understand and XPath is powerful, the tests are difficult to write and understand. The complete test condition has to be expressed in one XPath statement. Another option is Constraint Language in XML (CLiXML). CLiXML provides full first order logic to validate documents, whereas Schematron is restricted to boolean logic. A CLiXML document consists of a list of rules. Each rule specifies an expression which has to be fulfilled. It can be composed of 'forall', 'exists', 'equal', 'less', 'greater', 'and', 'or', 'not' and 'implies'.

For Verinec, CLiXML was preferred to Schematron for its more readable syntax and more expressive logic. Although CLiXML is the product of a single company, it has a well documented XML Schema defining its format and free validators may be implemented by others. This is exactly what has been done for Verinec, resulting in a open source CLiXML implementation [Jungo, 2005].

For testing the configuration, test cases as proposed in Section 5.5 can be used. Two kinds of test cases are distinguished. Static tests on the network configuration do not require a simulator, but specify requirements for the network. An example is provided in Listing 32. The test verifies whether each gateway specified in an interface definition points to an existing IP. Theoretically, a gateway could also be configured with DHCP, which would cause this test to create a false alert. However, gateways are almost always configured statically. Another example is looking for firewall rules specifying non-existing hosts. If the firewall specifically permits connections to an IP address (or address range), a network element with this address must be existing in the configuration. Otherwise, the rule is currently useless. If later a machine with that IP is added to the network, it is exposed to traffic.

Listing 32: StaticTestCase.clx

```
<forall xmlns="http://www.clixml.org/clix/1.0"
        var="gateway"
        in="//vn:nw/@gateway">
   <exists var="server" in="//vn:nw/@address">
       <equal op1="$gateway" op2="$server"/>
       </operator>
   </exists>
</forall>
```

The concept of rule checking is not limited to plain configuration mistakes. It can also include 'best practices'. If some configuration does not adhere to such practices, a warning is issued and solutions for the (potential) problem are proposed.

Other properties cannot be tested directly on the XML network definition. Expressing several different nodes and protocols involved in order to perform a service is too complicated in CLiXML. In that case, simulation can be used. The visualisation of the simulation can help to better understand why and at which point a test fails. The resulting event log can be analysed with CLiXML tests. The verification module does not explicitly support simulation test cases. But the simulation log can be saved, allowing to apply CLiXML tests to it. Section 10.2.6 outlines a possible implementation for automated test cases.

Each event in the simulation log contains XPath references to the configuration elements that caused it. This allows to identify the bit of configuration which was the cause for a test to fail. This is similar to the restrictors of the adaptation module. Again, if Verinec would contain a configuration editor, it could directly bring the administrator to the configuration part where the reason for the problem is located, for example the firewall rule which blocked a request.

8.8 Import Module

As stated in Chapter 7, many applications either only provide an overview of an existing network, but do not allow to configure it. Or they only allow to configure it, but provide no import function, requiring the administrator the design the network from scratch. VeriNeC aims at merging the two worlds. The means to configure the network have already been detailed in this chapter. This section is about the import module, which can create a base configuration from existing networks. The

network configuration can be manipulated after the import process to help where the automation failed to recognise all details. And of course, administrators probably want to change the network setup once they have the accurate current state represented in Verinec.

The network import is done in two phases, as discussed in Chapter 7. First, network traffic is analysed to determine the existing network layout and scan machines for known services and operating systems. The initial implementation was done in a bachelor thesis by [Aebischer, 2004] for traffic analysis and port scanning. [Seifriz, 2006] extended the functionality for topology detection in his bachelor thesis. The second phase is parsing configuration files to convert existing configuration into the Verinec format. This part has been implemented by [Antener, 2005] in her master thesis. It implements the use case from Section 4.2

8.8.1 Detect Existing Network Layout

Network detection is divided into four steps:

- Capture traffic (Jpcap)
- Analyse network structure (traceroute)
- Resolve host names (java.net.InetAddress)
- Port scan (Nmap)

Traffic capturing is done on the interfaces the user selects, either for a specified duration or until a specified number of packets has been captured. Traceroute, name resolving and port scans can be done in parallel, but to avoid confusing traffic capturing, it is done only after the first step is completed. Both traceroute and port scanning can be limited to specific subnets. This is most important for port scanning, as an administrator does not want to scan hosts that do not belong to her network. Scanning foreign hosts is at least rude – in some countries it is even considered illegal.

To capture traffic from the network, the interfaces have to be switched into the "promiscuous mode" to report all packets, as explained in Section 7.1.1. The Java environment does not provide this kind of low level access to sockets, but libraries using JNI to bind a native socket library can be used. A well known C library is libpcap [Jacobson et al., 2006]. In Verinec, Jpcap [Fujii, 2007] is used. This project wraps libpcap using JNI and provides Java classes to handle traffic capturing.[24] Both Jpcap and libpcap are available for Windows and Linux operating systems. Together, Jpcap and libpcap allow to open a network interface in promiscuous mode – when running Verinec with administrator privileges – and treat the received packets inside Java.

In Section 7.1.1, the problem of switched networks has been discussed. Traffic redirection was presented as an option to allow the sniffing host to see traffic destined to other machines. However, Verinec does not tamper with the network to receive more packets than the switches normally allow. Industry grade switches allow to mirror all traffic to a port for monitoring. For Cisco Catalyst switches, the feature is named SPAN (Switched Port Analyzer).[25] If really needed, it would be possible to run a program like `arpredirect` of the dsniff [Song, 2001] package from outside of Verinec to see more traffic. The author strongly recommends to discuss this with the network security responsibles beforehand and to do it only when service outages are of no concern.

Traffic analysis only tells which hosts exist somewhere in the network, but not how they are connected together. To uncover network structures, the traceroute method is used as described in Section 7.1.2. Three different possibilities have been implemented by [Seifriz, 2006]. Using the Java method Runtime.exec, the traceroute command of the operating system can be executed and its output parsed. As the information formatted for the human user, a parser is used to pick the IP addresses from the program output. Second, programs to perform a traceroute with the TCP protocol can be used both on Windows and Linux.

A last method is a custom ICMP based traceroute implemented by Seifriz. The custom traceroute allows for an optimisation to speed up the trace process. Verinec is only interested in the IP

[24] There exist two independent and quite different implementations for integrating libpcap into Java. Unfortunately both are named Jpcap. In Verinec, the implementation of Keita Fujii from University of Irvine is used. It seemed to be better maintained than the implementation by Patrick Charles, available at `jpcap.sourceforge.net`.

[25] The commands are described in detail on the Cisco web site: `www.cisco.com/warp/public/473/41.html`.

address of the intermediary hosts, while other traceroute implementations provide a small performance analysis from sending several (typically three) packets. After receiving just one response, the Seifriz-traceroute increases the TTL to proceed to the next hop. If no response is received before time out, there are still three attempts to get a response, after which the host is assumed not to respond. Sending ICMP packets and specifying custom TTL values again needs the low level access to the interface not provided by Java. Thus the Jpcap library was used for the custom traceroute too. For the timeout problem, a simple solution has been chosen. When four consecutive TTL values yield no response, the traceroute is terminated. Chances are high that the host was reached by then and is not responding to the traceroute packets. Just the four steps take quite a while, as for each TTL values, the timeout period of several seconds has to be waited out three times. The target host is appended to the generated list of four unknown hosts to inform the user what host was used. Although it is not obvious when looking at the network layout shown after sniffing, this situation can be distinguished from a couple of non-responding hosts between the Verinec machine and the target host. If there are less than four unknown nodes, the target was finally reached and responded. If there are four unknown nodes, the trace is terminated and no attempt to contact the fifth node is made. Whenever there are four unknown nodes in the result, this indicates that the trace was unsuccessful.

Listing 33: Nmap XML output

```
<host>
  <address addr="172.16.1.100" addrtype="ipv4" />
  <hostnames>
    <hostname name="veri-serv.verinec" type="PTR" />
  </hostnames>
  <ports>
    <extraports state="closed" count="1690" />
    <port protocol="tcp" portid="22">
      <state state="open" />
      <service name="ssh" product="OpenSSH" version="4.3"
        extrainfo="protocol 2.0" method="probed" conf="10" />
    </port>
    <port protocol="tcp" portid="80">
      <state state="open" />
      <service name="http" product="Apache httpd" version="2.2.3"
        extrainfo="(Fedora)" method="probed" conf="10" />
    </port>
    <port protocol="tcp" portid="443">
      <state state="open" />
      <service name="ssl" product="OpenSSL" method="probed"
        conf="10"/>
    </port>
  </ports>
  <os>
    <osclass type="general purpose" vendor="Linux" osfamily="Linux"
       osgen="2.6.X" accuracy="100" />
    <osmatch name="Linux 2.6.9 - 2.6.12 (x86)" accuracy="100"/>
  </os>
  ...
</host>
```

Port scanning has been discussed in Section 7.1.3. For port scans in Verinec, Nmap is used because of its maturity and easy integration. While there was at the time of this writing no JNI binding for Nmap the authors are aware of, Nmap can output scan results in XML format. There exists a Document Type Definition (DTD) for the output format, providing a precise documentation.[26] [Aebischer, 2004] implemented the scanning functions for Verinec. He executes the Nmap

[26] Available at `www.insecure.org/nmap/data/nmap.dtd`.

binary as an external process. The parameters tell it to scan the desired host and to output the result as an XML file. When Nmap terminates, Verinec parses the XML output and generates Verinec configuration from it. A sample output is provided in Listing 33. As Nmap has been compiled for all important operating systems, Verinec just includes several binaries for the different systems.

The interesting information for Verinec is the list of open ports. The result presented above was correct, although sometimes not completely precise. Port 443 is actually https served by apache - but apache uses OpenSSL . The OS guess is also quite good, the real version was 2.6.19.[27] It is newer than the Nmap version that was used in the scan. The Verinec node definition Schema for generic services is a copy of the format used by Nmap. All detected services (`port` elements) are copied into the node configuration as `generic-service` elements. Even if a service is supported by Verinec, it is up to the configuration parser or the user to build the exact configuration and create the service-specific element.

It would be possible to guess some more information about the network. For example on firewall configuration, networks a DHCP server is responsible for and so on. However, this would lead to unreliable information with a lot of manual cleanup required. As Verinec is intended to be used by the administrators of a network, Verinec uses administrator access to the devices in question. Parsing the configuration files of the services allows to import the exact configuration.

8.8.2 Configuration Import

Currently implemented are the import for the hostname, network interfaces in Red Hat Linux and for the Netfilter firewall framework [Antener, 2005]. Import of those informations can be achieved using three different options. The most simple case is importing configuration settings of the machine Verinec is running on. All that is needed is the root password for the machine, if Verinec is not running with root privileges. Given a host name and the necessary credentials for an administrator account, Verinec can also connect to a machine to import configuration from it. Here again, the J2ssh library is used to remotely execute commands or copy files to a temporary directory on the local machine for parsing. If connecting to a remote system is not possible for some reason, Verinec can also read configuration from files stored in the local file system. Where to retrieve or create such local files is described for each service in the following sections.

Hostname

Importing the name of the current machine is trivial on Linux machines. The command `hostname` without any parameters returns the desired information: the host name without domain. It returns the name configured on the machine, not any DNS name for the host IP. It is either executed locally or - for remote import - over ssh. If ssh is used, a DNS name or numerical IP is used to connect to the remote machine, but this is not necessarily equal to the host name.

Network Interfaces

To import the interface configuration of a Red Hat Linux system, Verinec scans the directory `/etc/sysconfig/network-scripts/` for files named `ifcfg-*`. For the loopback interface defined in the file `ifcfg-lo`, the user can choose whether to import it or not. To parse the files, Verinec employs the IniEditor library [Haldimann, 2005].[28]

Figure 8.14 shows the XML tree built for each physical interface. The mapping between configuration files and XML is straightforward. If a hardware address is specified in the base `ifcfg-eth`*X* file, it is put into the `hwaddress` attribute under `ethernet`. There is exactly one `ethernet-binding` child, as ethernet interfaces can physically only be connected with one cable. One `nw` element is created for IP and network mask configuration. Virtual interfaces are defined using files like `ifcfg-eth1:1` (interfacename:number), to have the device listen to more than one IP address. Each virtual interface file generates an additional `nw` element with the attributes specified by that

[27] The matching is based on behaviour characteristics of the network implementation. When there are no changes between different kernel versions, an exact match is not possible. The Nmap binary used in the test was published when kernel 2.6.12 was the most recent.

[28] The `java.util.Properties` class also uses INI-style configuration files, but it cannot handle files without section headings. The `ifcfg` files do not have sections, thus a third-party library had to be used.

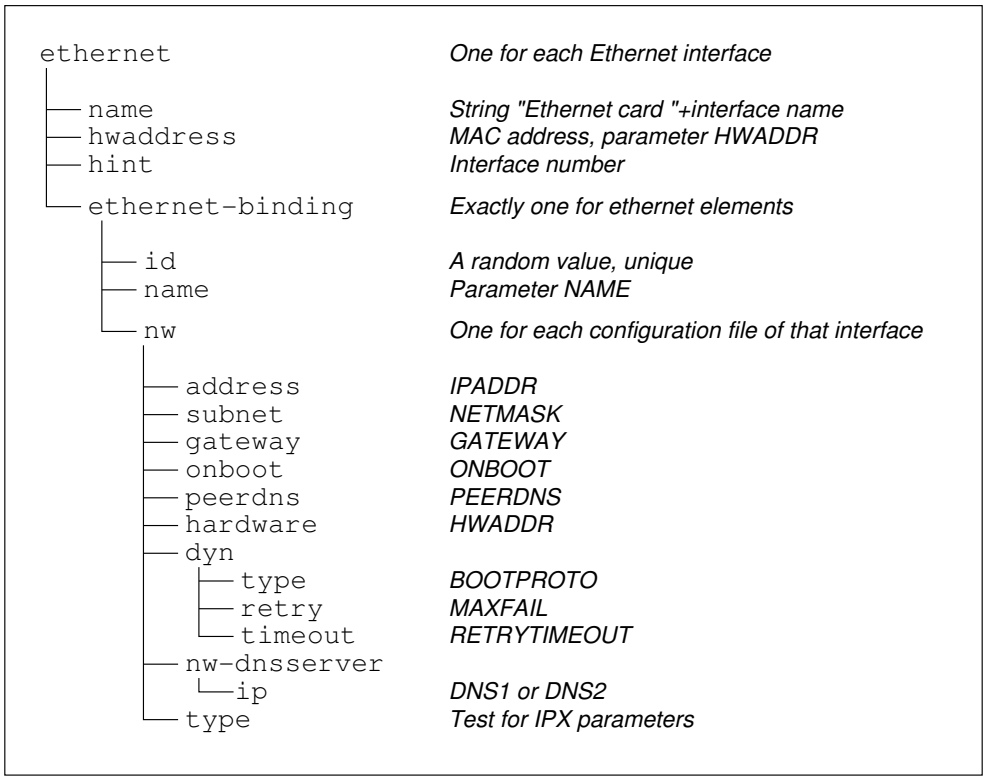


Figure 8.14: Information to generate the ethernet element.

configuration file. If virtual interfaces are defined, they result in having more than one `nw` element. The implementation has one shortcoming though. It relies on the filenames of virtual interfaces to reflect the interface names. If the virtual interface does not have the name of its parent interface in the filename, the mapping fails. However, standard convention for the filenames is to use the names as expected by Verinec. The example configuration from Figure 7.1 in Section 7.2.2 results in an XML fragment as shown in Listing 34.

Listing 34: XML fragment for an interface.

```
<ethernet name="Ethernet Card eth0">
  <hint system="pc" slot="0" />
  <ethernet-binding id="if1" name="eth0">
     <nw id="i1" onboot="yes" type="ip">
       <dyn type="dhcp" />
     </nw>
  </ethernet-binding>
</ethernet>
```

The Verinec importer can directly connect to the machine to import configuration from and copy the files to a temporary directory using ssh. If ssh access is not possible, the administrator may copy the complete network-scripts directory to the machine running Verinec and specify that directory to import the configuration from. The virtual interfaces provided an unexpected challenge for Verinec running under Windows. The Windows file system NTFS does not allow the colon character ':' in file names. J2ssh seems to silently ignore the errors, virtual interface definitions are not copied to the local machine. To handle this problem, the importer checks whether it is running on a Windows system. If the host system is *not* Windows, all ifcfg-* files are copied to a temporary directory using J2ssh's scp implementation. If the system is Windows, an ssh remote execute of `ls /etc/sysconfig/network-scripts/ifcfg-*` is done. Each file found that way is copied individually using scp. In single file mode, it is possible with J2ssh to specify

the target file name. The target filename is the source file, but with the colon replaced by an underscore.

Packet Filter

As explained in Section 7.2.3, packet filtering is used to protect a network. The configuration importer is implemented for the output of the `iptables` command of the Netfilter framework. The program output was preferred over the configuration files. The configuration files differ between distributions, whereas the program output is the same for all distributions. The output format of `iptables -Lvn` has been discussed in Section 7.2.3. To obtain all rules, the command **L**ist is used. The option **v**erbose tells iptables to show interface names and options, **n**umeric lets it display IP numbers and ports instead of hostnames and services. With the list command, iptables also displays statistics about how much traffic has been treated by each chain. Verinec has no use for the statistics, the parser skips this information and goes for the configuration information (default policy and the rules). As Netfilter inspired the Verinec XML Schema for packet filtering, the mapping is straightforward:

target: Target or chain to jump to if rule criteria match. Mapped to the packet-action-list.

prot: Protocol criteria to match. If tcp, udp or icmp, an according element is created, other protocols are ignored.

opt: Option for the packet, whether it is fragmented or not.

in: Interface the packet has to be received on. If test is present, map to match-in-interface if the interface exists in the configuration.

out: Interface the packet has to be going to be sent from. If test is present, map to match-out-interface if the interface exists in the configuration.

source: Source IP address of the packet. If not 0.0.0.0/0, mapped to the match-source child of match-ipv4.

destination: Target IP address of the packet. If not 0.0.0.0/0, mapped to the match-destination child of match-ipv4.

options: All other criteria of the Netfilter framework. Depending on the protocol specified in prot, DSCP, ECN, MAC, icmp, tcp, udp, reject-with and LOG are understood. Any other options are ignored.

If connecting to the remote machine to import settings is not possible, the stored output of the iptables command can be imported. The easiest way to get a file with the output is executing `/sbin/iptables -vnL > firewall.config`. The file can then be copied to the machine running Verinec, where it is possible to select it in the file chooser of the configuration import dialog.

Listing 35 shows an excerpt from the resulting Verinec configuration when importing the configuration shown in Section 7.2.3.

Listing 35: Firewall configuration fragment.

```
...
<vn:packet-filters global-in="chain222" global-out="chain222"
                   forward="chain333">
 <vn:packet-filter-chain name="INPUT" id="chain111">
  <vn:default-policy>
   <vn:accept-action />
  </vn:default-policy>
  <vn:packet-filter-rule>
   <vn:packet-match-list>
    <vn:match-ipv4>
     <vn:match-source>
      <vn:match-prefix address="192.168.199.3" />
     </vn:match-source>
    </vn:match-ipv4>
   </vn:packet-match-list>
   <vn:packet-action-list>
    <vn:drop-action />
   </vn:packet-action-list>
  </vn:packet-filter-rule>
 </vn:packet-filter-chain>
 <vn:packet-filter-chain name="FORWARD" id="chain222">
...
```

Chapter 9

Case Study

To illustrate the Verinec example, this appendix provides a case study of the different aspects of Verinec. A very simple network is imported, configured, tested and finally the updated configuration distributed.

The author built a small test network with real hardware in a laboratory of the university. The network consists of 6 client PCs and an intranet server, all connected with a hub. 3 of the clients are assigned static IPs, the other 3 use DHCP. The intranet server provides DHCP and DNS, as well as a web site. The network is the private class B network with number 172.16.0.0 and netmask 255.255.0.0. The clients with static addresses are named veri-staticX 172.16.1.X, the DHCP clients veri-dhcpX and DNS is set up to assign them172.16.1.1X. The server veri-serv uses IP 172.16.1.100. Connected to the same hub is also a Cisco Series 831 router for Internet access. Its internal interface is set to 172.16.1.254, which is also configured in each host as default gateway. Its WAN interface is connected to an Internet access provider. The internet side is simulated by one single computer with the name internet.net, which is enought to test the gateway.

9.1 General Operations

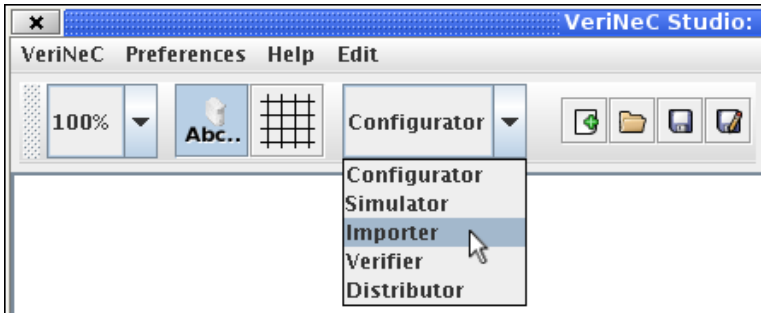


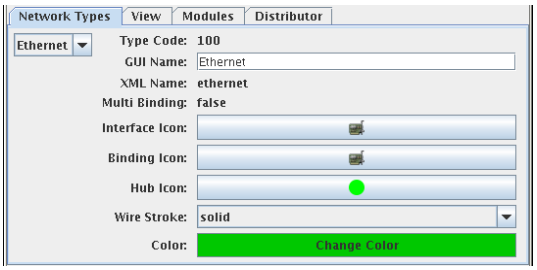
Figure 9.1: Main controls for the Verinec application.

When the Verinec application is started, a new, empty network is loaded. The icons of the main window, as shown in Figure 9.1, allow to (from left to right):

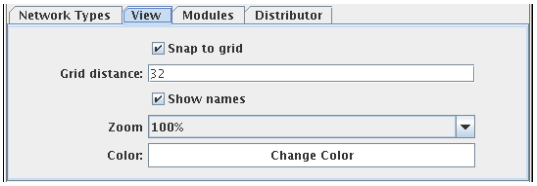
- Choose the zoom level, which is mostly useful to get an overview of larger networks.
- Toggle node description on and off.
- Show or hide the grid. If the grid is shown, nodes are automatically aligned to the grid lines upon move or creation.
- Choose the module to be used in the drop-down.
- Create new network.

- Open a previously saved network.
- Save the network.
- Save the network under a new name.

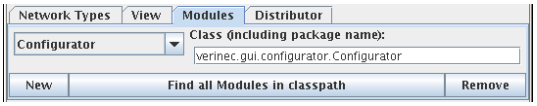
Loading and saving can also be performed through the menu labelled “VeriNeC” or with the typical keyboard shortcuts ctrl-n(ew), ctrl-o(pen), ctrl-s(ave), ctrl-shift-S(ave as). The “Preferences” menu contains one entry named “Options”, which brings up a dialog to configure various aspects. Each aspect is grouped in a tab. After the Modules tab, the configuration panel for each module is added. As mentioned in Section 8.1.1, a module can provide a configuration panel, but does not need to. The Configurator module is the only one providing a configuration dialog.



Network Types allows to customise colours of wires and icons for interfaces and bindings.



View allows to change the defaults for zoom level, name display and snap to grid. The background colour of the draw area can also be modified.



Modules is used to manage the available modules. To add new modules, the button “Find all Modules in classpath” launches a process looking through all classes for those implementing the IVerinecModule interface. All new modules are added to the application.

The Distributor allows to specify a directory in case translated configuration should be stored on the local machine instead of being automatically distributed.

The “Help” menu finally contains only an “About” entry to bring up a window telling what Verinec is and who developped it.

9.2 Import

The importer module was first put to examination. To import a network, the “Importer” module has to be selected. The menu bar now shows an entry labelled Importer, containing the entries “Start Network Analysis” and “Read configuration files”. Choosing the network analysis allows to start traffic sniffing to detect the layout of an existing network.

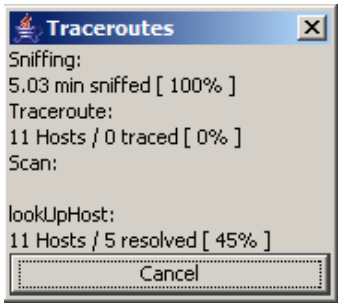


Figure 9.2: Progress dialog.

Verinec was running on a laptop connected to the hub of the test network for five minutes. During that time, the computers where pinging each other, ssh connections opened and web pages retreived. The capturing worked fine and produced the network shown in Figure 9.3. If the hosts do not show up in the view, it helps to zoom out to locate where they have been placed. The same result can be seen in Figure

9.4, but has been arranged by hand to improve readability. All hosts existing in the network have been detected. The machine running Verinec is shown too, to indicate from which point the network has been analysed. Because of the hub, most hosts are directly connected. 63-236-73-147 is the Internet server, reached through the router. Its name is the converted IP number, because reverse DNS was not configured for it. There is one notable problem and one mistake.

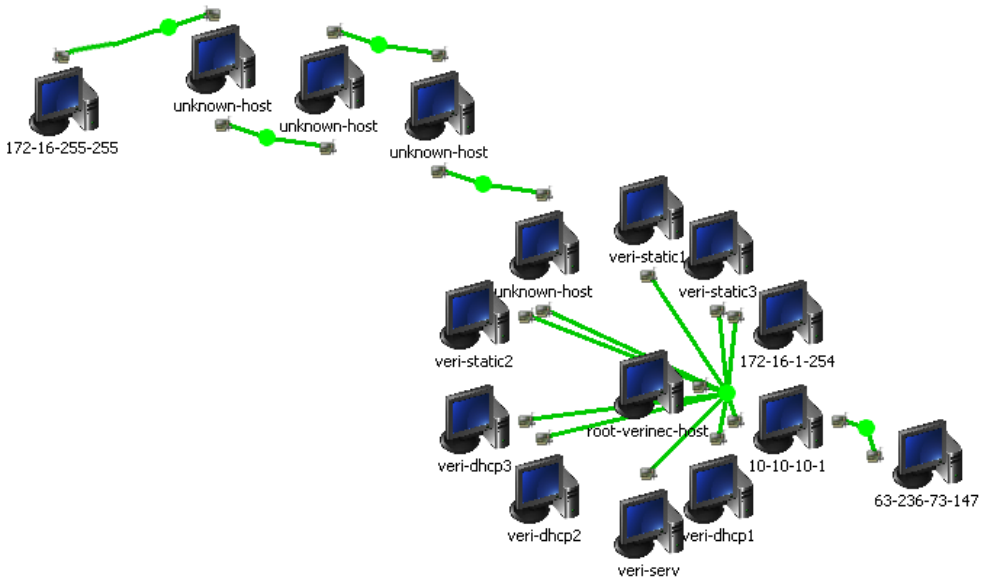


Figure 9.3: After traffic analysis.

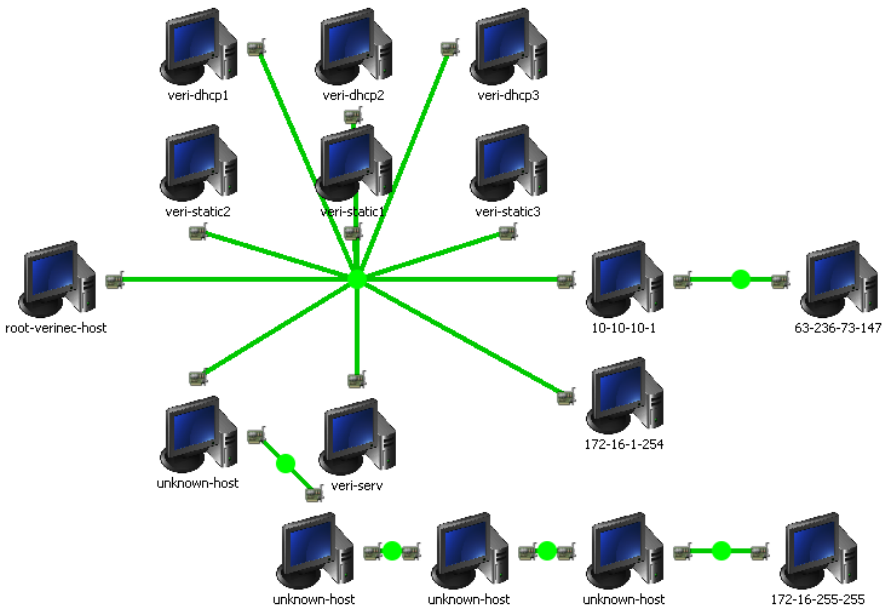


Figure 9.4: Rearranged layout to improve readability.

The problem is that the gateway has been duplicated. Although the Cisco router was set up to use 172.16.1.254 for its internal interface, the Internet seems to be connected through a server with IP 10.10.10.1. The reason for the confusion is that the time exceeded message from the Cisco router still used the IP of 10.10.10.1 instead of the one the router was set to. Examining the traffic, it would be hard to determine that this is the same device. In this situation, looking at the MAC

address would solve the problem, but as soon as such a device is not the first in the line and thus connected to a different segment, the MAC address can no longer be used.

The error is that the broadcast address 172.16.255.255 had been mistaken for a normal address. Although in most cases, an address ending with 255 is a broadcast address, this cannot be taken for sure. It is up to the human operator to detect and remove the address. The situation also illustrates the output when no response on the traceroute is received. To correctly interpret the result, we need to remember that the maximum number of non-responding hosts is 4. As there are four unkown-host nodes, the trace was interrupted, and the final target added for information. It might well be closer to the host running verinec than the four nodes shown – or even further away. Would the target be an existing host in network 172.16, the most probable explanation is that either the host does not respond to the trace packets, or a firewall is silently dropping packets or responses. If there are only three or less unkown nodes, the meaning is quite different. The last host in the line is separated by exactly this number of nodes from the sniffing machine, as from that TTL on, it replies to the packets, altough intermediate stations do not report when they dropped a packet due to TTL dropping to 0.

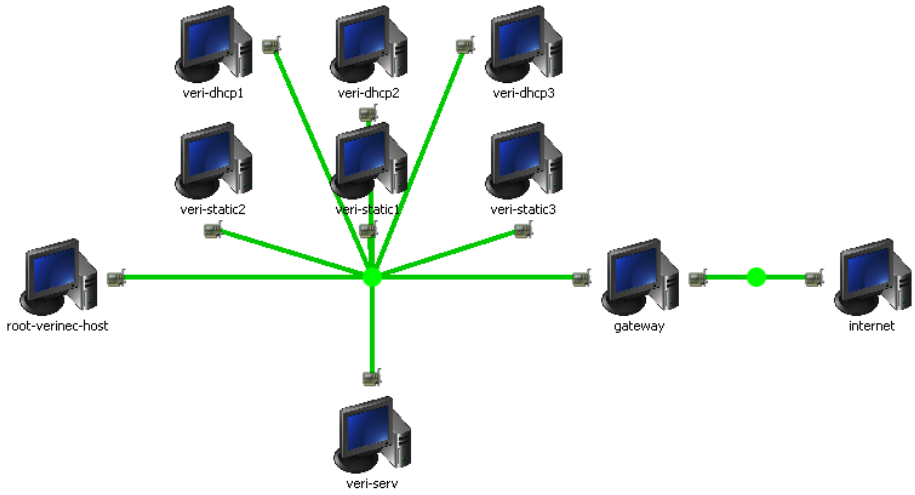


Figure 9.5: The final network with the incorrect hosts deleted.

Cleaning up the problems and errors of the network in Figure 9.4, the user should get to a network as shown in Figure 9.5. The gateway has been reduced to one single node, the broadcast address is deleted and the Internet host is named “internet”. The verinec-root-host has been left to facilitate orientation, although it normally will not be part of the network.

> **Warning:** All techniques used in the import module can collide with network usage regulations inside an institution. For port scanning, particular care is required. The author recommends to only test this feature in experimental networks or when explicitely granted permission from network responsibles.
> The initial scanning phase gathers traffic from the network. This does not generate traffic, but can still be forbidden by usage policy.

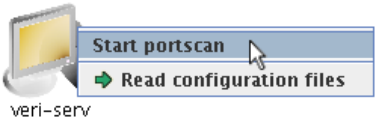


Figure 9.6: Start the port scan.

One more kind of information can be gathered: the services running on a host. A click with the right mouse button on a host pops up the context menu shown in Figure 9.6. After a last question to confirm whether the user really wants do a port scan, the Nmap application is launched. It is told to scan the host with the selected name and to output the scan results in XML. The scan is run in the background. Information is added to the node after it finished, and the user is informed of the successful port scan.

On the first try, scanning `veri-serv` took a very long time and produced no information at all. It turned out that iptables successfully blocked the scan attempt. After running the

command `/etc/init.d/iptables stop` on the machine, the scan produced a more interesting result and took only a couple of seconds. The output file is parsed and integrated into the Verinec configuration. Currently, all services appear as `generic-service` elements. Listing 36 shows a snippet of a resulting configuration. The importer could be improved to use the correct service element instead of `generic-service` if there is one available in the Verinec configuration. The service 'domain' for example would be represented by the `<dns>` element.

Listing 36: A sample node

```
...
<generic-service protocol="tcp" portid="53">
  <port-state state="open" />
  <port-service name="domain" method="probed" conf="10" />
</generic-service>
...
```

The port scan of the machine veri-serv reveals that there are many services running. Table 9.1 lists the services. The administrator should close all services that are not really need. 'vmware-auth' for example remains from a vmware installation that was previously done, but later removed. It is currently not even used, but presents a security risk.

22	ssh
53	domain
80	http
111	rpcbind
443	ssl (https)
825	status
904	vmware-auth

Table 9.1: Open ports on the veri-serv machine.

9.2.1 Import configuration information

The structure of the network has been established in the previous section. Now it is time to parse the configuration of a machine. Clicking the second option labelled "Read configuration files" in the menu from Figure 9.6 pops up a dialog to import configuration. This dialog is shown in Figure 9.7. As has been mentioned in Section 8.8.2, import has only been implemented for a couple of services running on Linux. The dialog contains three areas. At the top, the user can choose whether to import configuration from the local machine, from a remote machine over the network or from configuration files. The third option is a last resort if connecting to the remote machine does not work and one has to copy the configuration to a portable media or send from the remote machine. The middle area lists all services and allows for each of them to select whether to import it or not. Importing the host name is useful as the DNS entries do not necessarily match the internal name of the host. For Ethernet, the location of the configuration files can be changed if needed. The iptables command can also be modified - but if the parameters change the output, the parser might no longer work correctly. The bottom part is used to specify the connection to the remote host. Host can either be a DNS name or an IP. In the test, no ssh key file has been used, so the keyfile and passphrase fields remain empty.

The configuration of `veri-serv` was imported to test the features. Importing the Ethernet interfaces produced two warnings but otherwise got all information correctly from the remote machine. Red Hat used two settings named `userctl` and `ipv6init` which are not known by the importer. User control tells whether a normal user has the right to bring the interface up or not. The prefix ipv6 indicates the setting is related to IP version 6, which is not supported by Verinec anyway. Distributing the configuration built exactly the same configuration files as the original import, except the two fields userctl and ipv6init are omitted. One minor problem is that existing

interfaces are never replaced. The new interfaces are added to the configuration even if an interface with the same name already existed. We did not want to risk overwriting existing configuration and prefer to leave merging to the user.

Iptables import worked almost perfect. As the firewall had been disabled for the Nmap port scan, `/etc/init.d/iptables start` had to be called before running the import. The only warnings are about the protocols 'esp' (Encapsulating Security Payload) and 'ah' (Authentication Header). Both protocols are related to the IPSec standard. The firewall would accept all packets belonging to those standards. In the Verinec packet-filter configuration, those protocols are not supported and thus cannot be imported. Contrary to the interfaces, the packet-filter is replaced if it is already defined on the node. The XML Schema allows only one packet-filter element.

Both for Ethernet and iptables, there is a report dialog to tell about problems on import, as shown in Figure 9.8. It also shows the original configuration to facilitate locating the source of import problems. The output can also be saved to a file for later use.

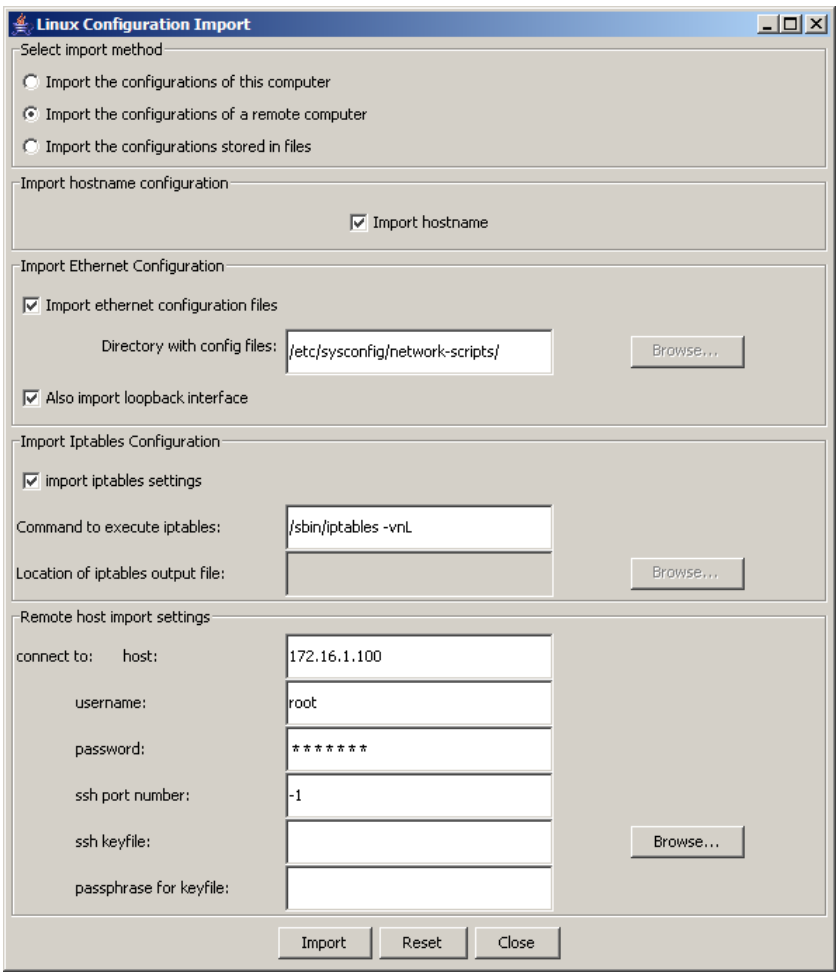


Figure 9.7: Dialog to configure Linux configuration import.

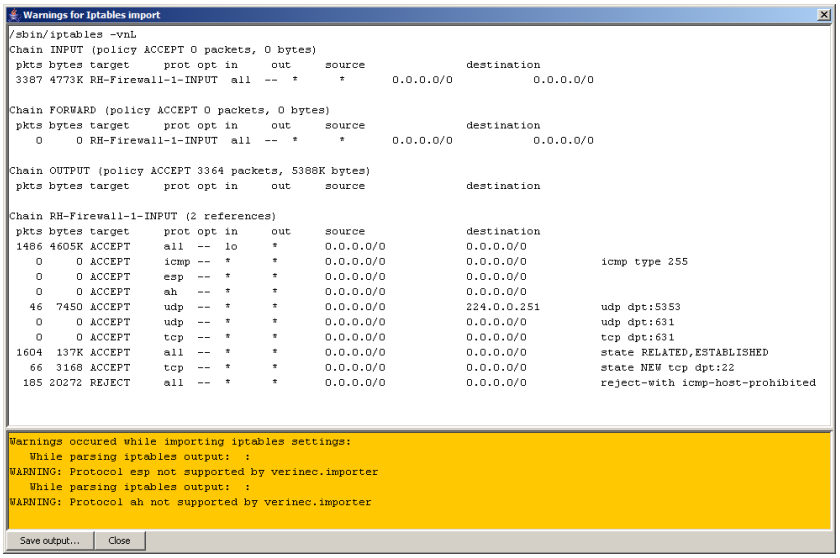


Figure 9.8: Report of iptables import.

9.3 Edit

Figure 9.9: Configurator toolbar.

To change the configuration, the "Configurator" module has to be selected. A new tool bar, shown in Figure 9.9, appears in the main window. The arrow signifies that nodes can be moved and interfaces connected. After selecting the computer icon right to the arrow, new nodes will be created on left mouse click.

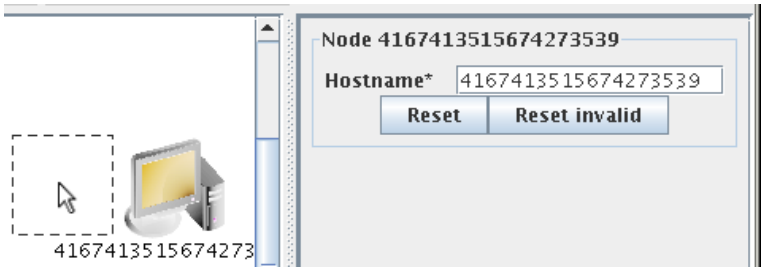


Figure 9.10: Create new nodes.

Figure 9.10 shows the node creation. A frame indicates where the new node will be placed. Clicking the mouse creates the node. The editor dialog on the right side now shows the new node. The name is just a random number, and the node has no services or interfaces. It can be changed in the Hostname field of the editor.

Adding interfaces is done by right clicking on the node where the interface should be added. A context menu pops up to choose the type of interface, as shown in Figure 9.11. This works both in move and create mode. When interfaces are added, the node editor on the right side allows to edit properties like IP or DNS servers, as shown in Figure 9.12. Besides adding interfaces, the context also allows to delete the node. The "Properties" entry lets the user choose a different image to represent this node.

Figure 9.11: Adding interfaces.

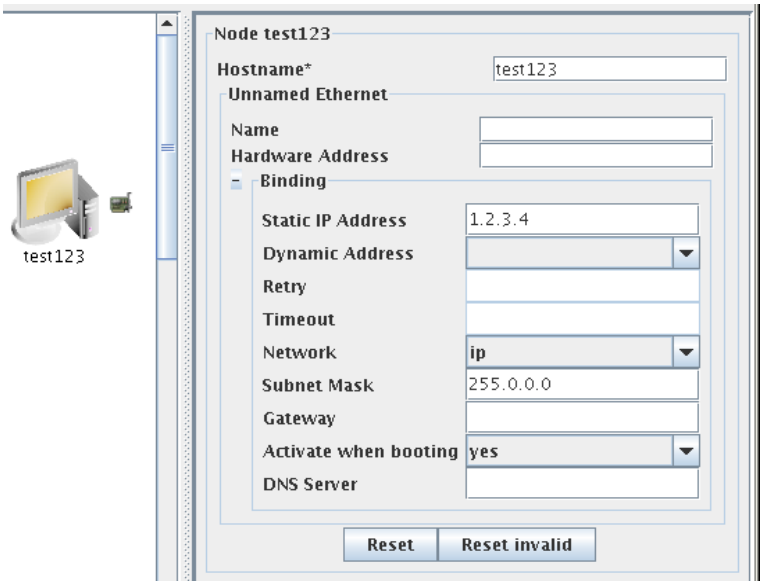


Figure 9.12: Node editor.

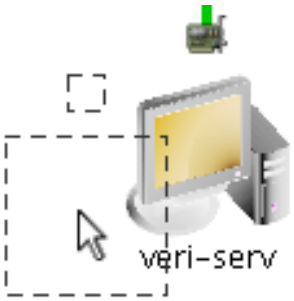


Figure 9.13: Move node.

The configurator stays in creation mode until switching back to move mode. After changing back to move mode, nodes can be moved by clicking on them with the left mouse button and dragging them to the new location, keeping the mouse button pressed. A frame indicates the new position, as illustrated in Figure 9.13.

Network connections are also created by dragging the mouse. Connections are always defined on the binding level, by dragging the mouse from one icon to another (see Figure 9.14). Connecting two bindings is only possible if both are of the same type. To add more bindings to the same hub, click on the binding and drag to the hub to connect to. The other direction is not possible, as dragging a hub moves the hub around instead of creating a new connection.

Figure 9.14: Connect two bindings of type Ethernet.

Network types allowing only one binding per interface show the binding icon. The others show one icon for the interface and a queue of bindings, for example the WLAN interface of node `test456` in Figure 9.14. The context menu of the interfaces allows to set the side of the node the interface should be shown and to delete it. The context menu of multi-binding interfaces additionally allows to add new bindings. The bindings context menu allows to deconnect the binding from its current hub or to delete the binding completely.

Multiple nodes can be selected at the same time. A selection is made by dragging the mouse from an empty place over the nodes to select, as illustrated in Figure 9.15. Individual nodes are added to the current selection by holding the `control` key pressed while clicking them with the left mouse button. If the node is already selected, it becomes deselected. Moving the selection is done by dragging any of the selected nodes, deleting by pressing the `delete` key or using the context menu of any of the selected nodes. The “Edit” menu in the main menu bar allows to manipulate the selection. “Select all nodes” or ctrl-a marks all nodes of the network as selected

and “Deselect all nodes” or ctrl-shift-a removes the selection. “Invert selection” or ctrl-i allows to select all unselected nodes and to deselect all select nodes.

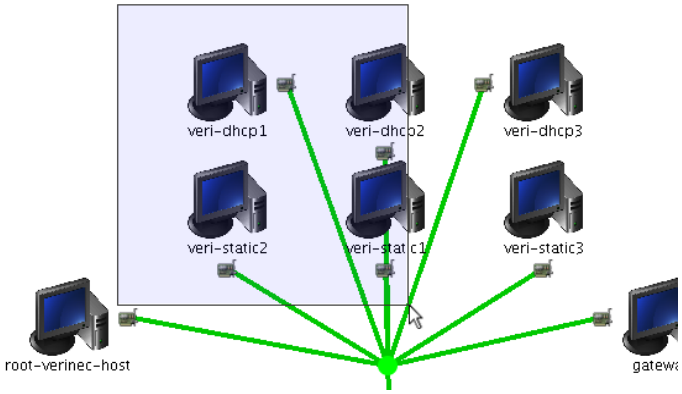


Figure 9.15: Select multiple nodes by dragging the mouse over them.

The editor allows to configure devices of any type. To illustrate this, compare the Verinec interface configuration editor from Figure 9.12 with Figure 9.16. Each time, the same configuration is specified, but the interfaces are very different.

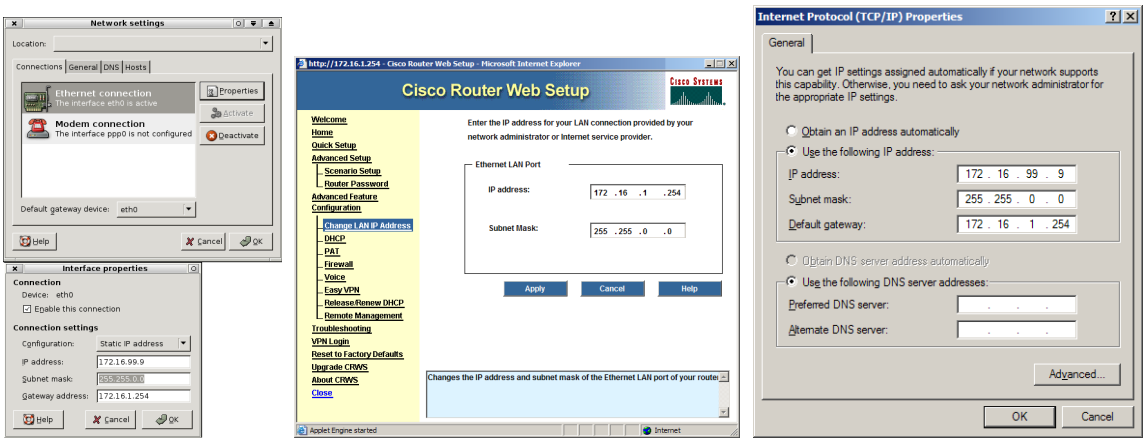


Figure 9.16: Different configuration interfaces for the same task. From left to right, the interfaces are from the KDE window system for Linux, a Cisco web interface and from Windows XP.

9.4 Simulation

The verification module is loaded by choosing “Simulator” in the module chooser. In the simulation mode, the node context allows to add events to nodes, as illustrated in Figure 9.17. The right part of the GUI shows a timeline with all input events, as shown in Figure 9.18. Events can be selected to edit them later or delete them.

Figure 9.17: Create input event.

Each event has a time, expressed in the simulation time. This time is abstracted, it is only used to define the order the events occur in. The type of the event is chosen in a drop down. For now, only two applications exist. “Ping (TCP)” tries to establish a tcp connection with the destination host. “Ping (UDP)” sends a udp packet and expects a packet to come back from the destination host. Both have their daemon counterparts, as described in Section 8.7. The source field is the host the event was created on. Destination is the IP of the target node of the command.

The input events can be saved to a file and restored later using the first couple of entries in the "Simulation" menu. The current network name is saved with the events. To load the events, the same network has to be loaded first, otherwise Verinec refuses to load the input.

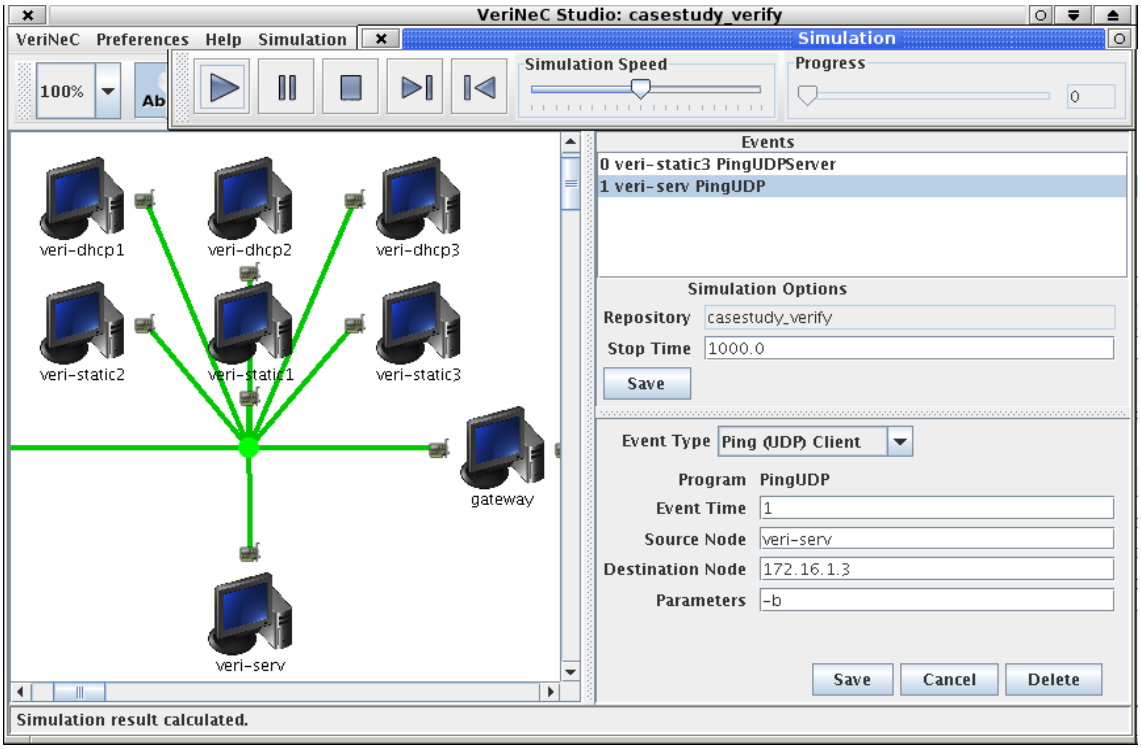


Figure 9.18: Input events for the simulation.

Once there are input events available, the "Run Simulation" command becomes enabled. It starts the simulator that calculates the result for the current input events. When it is finished, the play button in the simulation icon bar can be pressed to visualise the simulation. The simulation animation is controlled with the five buttons to start, pause or stop the simulation and to jump to the next or the previous event. A slider is used to set the speed of the animation, a second slider to select the current time.

The menu also allows to save the simulation result on file. As the input events, this file follows the events XML Schema. A saved simulation result can be loaded with the last menu command to play the animation later on. The saved result can also be used together with CLiXML tests to verify the correctness of the network, as described in Sections 8.7.2 and 10.2.6.

9.5 Adaptation

The distributor module allows to run the adaptation process. It is possible to distribute the configuration of the whole network using the disk icon in the distributor icon bar (see Figure 9.19). The second icon allows to export the translated configuration to files for manual distribution or debugging. If only the configuration of one node is to be updated, the context menu of that node can be used to start the distribution.

Figure 9.19: The distribution icons.

All nodes need a nodetype declaration for the distributor module to work. Nodetypes cannot be edited with a GUI, but have to be written in the XML source code. This is described here for the example network used in this Chapter. The network is stored in the directory `data` in the working directory of the application. It contains the file `globals.xml`, where the `typedef` section has to be added as shown in Listing 37

Listing 37: globals.xml

```
<tr:typedef>
 <tr:type name="redhat" id="redhat">
  <tr:service name="ethernet" translation="linux-redhat"
              target-id="distlinux"/>
  <tr:service name="packet-filters" translation="iptables"
              target-id="distlinux"/>
  <!-- add more translators here -->
 </tr:type>
 <tr:type name="cisco" id="cisco">
  <tr:service name="ethernet" translation="cisco800S"
              target-id="distcisco"/>
  <tr:service name="packet-filters" translation="cisco800S"
              target-id="distcisco"/>
 </tr:type>
 <tr:target name="distlinux" id="distlinux">
  <tr:scp username="root" password="verinec" host="{!hostname}"
          hostVerification="no"/>
 </tr:target>
 <tr:target name="distcisco" id="distcisco">
  <tr:cisco><tr:snmp targetAddress="172.16.1.254">
   <tr:security><tr:community snmpversion="v2c">
    private17_veri
   </tr:community></tr:security>
  </tr:snmp></tr:cisco>
 </tr:target>
</tr:typedef>
```

The node definitions are located in the `nodes` directory inside `data`, each in a separate file with the name of the node. Each node has to be assigned the correct node type. For the Linux machines, the line `<tr:nodetype type-id="redhat"/>` is used. The gateway is of type `cisco`. `root-verinec-host` and `internet` should not be translated. The type `no-translation` is used as described in Section 8.4.1. Starting the translation on node `veri-static1` first lets the restrictor do its work. A warning dialog as shown in Figure 9.20 reports on the problems encountered. Here, three DNS servers exist in the configuration of an Ethernet card, but Red Hat only supports a maximum of two servers. This is also reflected by a line in the resulting configuration:
`#Not more than 2 dns servers supported (DNS3=63.236.73.147)`
The configuration is either distributed directly or written to a file in a temporary directory, depending on whether distribution or export have been selected.

Translation and distribution on the test network was no problem. Ethernet configuration remained unchanged. Some differences from the configuration originally imported where observed for iptables. Verinec uses a fixed order, while iptables is more flexible. The order of chains in the ouptut of `iptables -vnL` depends on order of the input commands. In Verinec, the chains are ordered by references. Some option tests can be specified in any order. The parser generates Verinec configuration in the correct order, independent of the input order. After the export, the order always follows the XML structure. But both configurations are equivalent. For example, the tests `tcp dpt:22 state NEW` and `state NEW tcp dpt:22` match exactly the same packets: connection set up attempts on port 22 (ssh).

After the first import-adaptation round, the parser can import the configuration without any warnings. The order of tests and chains is now also stable and remains in the same order regardless how many times the import and adaptation process is repeated. This is also due to the trick described in Section 8.5.3. If the default chains would use jumps, one additional jump would be added on each import-adaption round, making the configuration less readable.

A small packet-filter configuration has been added to the `gateway` to have something to distribute for Cisco. It is the same as the one used on `veri-serv`. For iptables, it was translated without any warnings. Because of the differences between iptables and Cisco, the Cisco restrictor

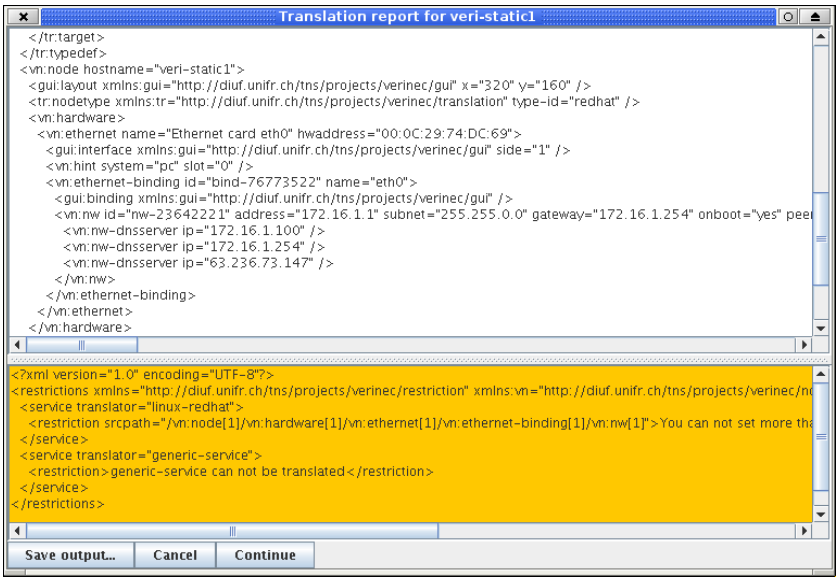


Figure 9.20: Report of Ethernet export on veri-static1.

generates a couple of warnings. One is about Cisco not having a concept of “Input” and “Output” chains. The other is for the ICMP type on reject that Cisco does not allow to specify. Listing 38 shows the resulting restrictor report.

Listing 38: Restrictor report for Cisco

```
<?xml version="1.0" encoding="UTF-8"?>
<restrictions>
 <service translator="cisco800S">
  <restriction srcpath="/vn:node[1]/vn:hardware[1]/.../vn:nw[1]">
   No support for onboot argument on Cisco
  </restriction>
 </service>
 <service translator="cisco800S">
  <restriction srcpath="/vn:node[1]/.../vn:packet-filters[1]">
   Be aware that for cisco, there is no notion of forward chain.
   Packets always pass input and output chains of their interfaces.
   The translator ignores global-in and global-out, adding only the
   forward chain to both in and out direction of each interface.
  </restriction>
  <restriction srcpath="/vn:node[1]/.../vn:reject-action[1]">
   Filter-chain RH-Firewall-1-INPUT-gw: Not possible to specify ICMP
   type or code on reject action.
  </restriction>
 </service>
</restrictions>
```

Chapter 10

Conclusions

Network management has many different aspects. This thesis has focused on configuration management. In the conclusion, I shall recapitulate on the achievements of the Verinec project and outline possible future work on both theoretical and practical problems.

10.1 Achievements of the Verinec Project

In this project, different aspects of network configuration management were studied and combined in a network configuration system. We implemented a **working prototype** to illustrate the concepts. While still far from being ready for productive use, this application demonstrates the potential of our ideas. It can help an administrator to keep track of a network, and to import, manipulate, verify and distribute configurations. The achievements realised by the prototype indicate that our propositions for network configuration management are realisable.

- Conclusion: The configuration of network services can be abstracted from specific implementations and can be represented with XML.

Abstracting configuration from specific implementations is a fundamental requirement for configuring heterogeneous networks. It is not practical to re-implement verification and network simulation for different implementations of the same service, simply because of varying configuration formats. An abstract representation of service configuration can be established, since each service is defined by a protocol. This limits the number of options that must be configured and permits the representation of similar features in a uniform way.

We also implemented ways of discovering the structure of networks and of parsing the configuration data of selected services. Analysing existing networks has been done before; including this function does, however, make the Verinec prototype more complete. With the import features, Verinec can be used to improve an already existing network. Generating XML configuration from implementation-specific configuration files and from gathered network traffic was quite successful.

- Conclusion: Automated translation into implementation-specific formats is possible. The distribution of configuration can be achieved with standard remote access protocols.

Although different implementations of a service follow varying philosophies for their configuration formats, it is possible to generate their configuration from the same XML data. A well-designed XML format for services provides enough information on the intent of configuration to produce configuration files according to very different of syntax rules. Non-filebased configuration means like WMI or SNMP are equally easy to support.

Some of the work involved in implementing the distribution of translated configuration to the devices could have been avoided by using the open source Java build tool `ant`[1]. Unfortunately, the ant developers added support for ssh and secure copy only after we started working on Verinec. The parameters are very similar to those used in Verinec. This not only confirms that our implementation is reasonable, but also means that rewriting the distributor to use ant would be

[1] See `ant.apache.org`.

straightforward. The distributors for Cisco devices and WMI could have been implemented as ant tasks, instead of creating a complete distribution framework specially for Verinec.

- Conclusion: Configuration can be verified with network simulation and test cases.

Network test cases proved to be a useful tool, allowing one to specify the requirements of a network in an intuitive way. The strength of simulation tests for network verification is flexibility. They can describe *what* the network must do, without having to make assumptions about *how* the components must interact in order to achieve this. This is best illustrated with an example. Consider the use case presented in Section 4.5. The administrator split the workstation role into separate roles, namely "Safe Workstation" and "Power Workstation". Some workstations need full Internet access (Power Workstation), while others are limited to HTTP (Safe Workstation). The different roles use tests that have to be satisfied to define their requirements and restrictions. To successfully run the tests associated with the Power Workstation role, the administrator can change the intranet firewall configuration. A rule has to be added matching the Power Workstation IP range and allowing full Internet access, while a rule for the Safe Workstation IPs would only permit the HTTP protocol. The administrator could also choose a different setup and forbid all Internet connections for Safe Workstations and configure them to use a HTTP proxy on a node allowed to connect to the Internet. For either solution, the tests do not need to be adjusted simply because different setups are used to fulfil the requirements. Both solutions provide the same service, which can be validated with an identical test - one of the benefits of using simulation to validate the requirements.

While studying the subject of simulation, we also noted that simulation can be used for tests that are completely impossible in the real world. The failure resilience of networks can be tested by repeatedly running the simulation tests with different nodes switched off. This procedure is described in Section 5.6 on testing redundant backup strategies.

10.2 Future Work

In its current state, the Verinec application is not suited to productive use. Our goal was to write a prototype application in order to experiment with the concepts presented in this thesis. The prototype was successful, but before a usable product could be built, a lot of work would still be needed. The configuration definition format itself could be improved. This is discussed below with excerpts from two papers the author presented at conferences. Verinec only supports an exemplary range of services. In order to make it usable, more service definitions, translators and importers would be needed. Verinec should be able to store older versions of configuration and track changes. For larger systems, authentication and authorisation should be implemented.

Other areas of network management such as monitoring or handling alerts should be supported if Verinec is to become a complete management system. As there are already tools that perform these tasks, the best solution would be to integrate Verinec into an existing network management solution or, inversely, to integrate existing tools into Verinec.

10.2.1 Role Model

This section is based on the paper [Buchmann et al., 2007b]. The observations in Section 4.5 point out that nodes play distinct roles within networks. These roles could be modelled by the Verinec configuration. A role defines requirements for a node and provides configuration templates that incorporate these requirements. A role model not only helps to avoid redundancy, but also deepens the knowledge of the network.

The concept of the role model was sketched in the use case in Section 4.5. Nodes in a network can usually be classified into different types, such as managed switches or application servers. Network planning defines the policies valid for each type of node.

One goal of the role model concept is to ensure that network policies are respected by the configuration. To achieve this, each role contains a collection of test cases, all of which must be satisfied by a node taking on a particular role. The tests are a good instrument for defining, in a precise *and testable* manner, what a node must be able to do and what is forbidden (see Sections 8.7 and 10.2.6 for more on the test cases).

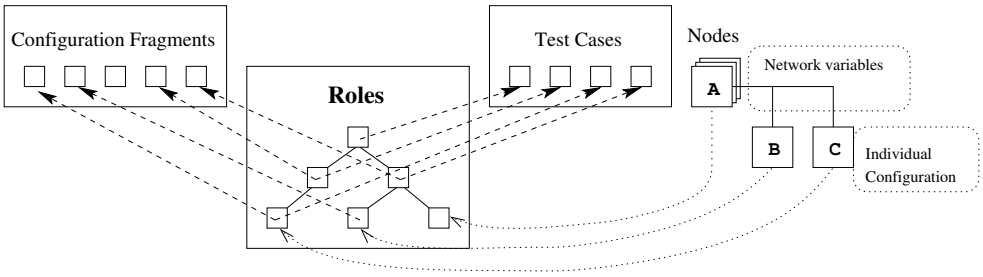


Figure 10.1: Roles function as a layer of abstraction between nodes and policies.

Configuration declaration itself should not be redundant. Configuration fragments for specific tasks are stored in a template library. The roles specify the configuration fragments to be used for nodes in that role. The fragments can refer to parameters defined by a role, by the node or the subnet the node is in. Variables (see Section 8.2.3) can be used to customise configuration. A more elaborated concept of adapting template configuration to its context would be environmental acquisition, which is discussed in the next section. The actual configuration of a node is created from the fragments specified by its role, and, if needed, with individual configuration. In addition to the parallels illustrated in Section 5.1, there is another analogy between configuration data and programming languages. Programming languages use functions or classes to structure code and avoid redundancy. With configuration data, templates can be employed for the same purpose. Figure 10.1 shows the relationship between networks, nodes and the role hierarchy, as well as the links from roles to test cases and configuration fragments.

A different role hierarchy can be created for each project. The configuration fragments and test case libraries, on the other hand, are best kept global to avoid the redundancy of having several copies of a template. An error in a configuration fragment could then be fixed in one place, being immediately visible in all roles using that fragment. As an example, let us take a firewall configuration that rejects all IP packets violating the standards. Should an administrator one day discover that she forgot to include a particular type of packet, she can simply add another rule to the fragment. All firewalls using this configuration fragment will then use the new rule.

If there is a limited number of nodes that perform the same task, assigning them the same role will be enough. However, if 50 or 100 computers have an identical setup, assigning roles will not be sufficient. To solve this problem, a *node group* can be designed to specify the setup for an arbitrarily large set of nodes. A node group defines the base configuration (role, number of network interfaces, additional configuration) and can specify individual parameters for each node in the group as needed, for example MAC addresses. This also permits economy in terms of the tests Verinec has to run: it might be enough to test one node in the group to see whether the network is set up correctly.[2]

Verification and role model

The role information can also be used to verify configuration. [Jungo et al., 2007] distinguish two kinds of test cases. A *positive test* requires that something works, a "must" rule in the policy. A *negative test* is successful when an operation is prevented by the network, and is thus a "must not" rule.

The root role defines the maximum permissions any node within the network can have. Thus, the root role must specify all positive tests for things that should be allowed, and can list negative tests that hold true for every node in the network. It is forbidden to add positive tests when subclassing a role, they can only be omitted or else replaced by a narrowed test. The requirement "Internet access" could, for example, be replaced by a requirement "HTTP access". Positive tests are only inherited if this is explicitly specified. This avoids inadverted changes to other roles when one adds a new positive test to the root of a chain of roles. Negative tests are always inherited, one cannot get rid of them; they can only be replaced by more restrictive tests.

[2] However, the information loss if not every node is tested might result in errors. Think of a firewall rule blocking out just one IP number or MAC address of one of the nodes in the group.

The configuration fragments are inherited from role to role as well. However, they can be removed; new fragments can be specified and individual configuration can be added. There is no need for restrictions concerning what configuration can be changed or overwritten, since the policy is enforced by the test cases.

10.2.2 Environmental Acquisition

This section is based on the publication [Buchmann et al., 2006]. Class inheritance is an important concept for proper code design, and helps avoid redundant definitions. It forms a hierarchy of *"is a"* relations between types. Type inheritance is not only used in object-oriented programming languages, but in other areas as well. In network configuration, a node can be said to be *of a certain type*. It may be, for example, a workstation or a firewall appliance. There can be specialisations for distinct requirements on hard- and software, such as a graphics workstation or a web developer workstation. In Section 10.2.1, this concept has been named the *role* of a node.

However, the "is a" relation of a class hierarchy is not sufficient for all kinds of information models. Many objects also have a *"is in"* hierarchy, that is, their *context* is relevant for their behaviour. Taking the context of nodes into account can improve network configuration management to a great extent. Networked devices are located *in a room*, appear *in the subnet* and so on. The contexts define which printer to use by default, as well as the network gateway and many other configuration items. [Gil and Lorenz, 1996] represents *environmental acquisition* as a concept for handling context. The idea of acquiring information from the containment hierarchy has been implicitly used in many places. Gil and Lorenz propose to explicitly model acquisition in the programming or configuration language.

There has been one previous attempt at defining a network configuration language with explicit environmental acquisition that the author knows of. The configuration description language "Anomaly" was developed to explore the possibilities of environmental acquisition for network management by [Logan et al., 2002]. It is a simple configuration language that defines properties and uses them in the style of logic programming. The language is less flexible for incorporating new services than the Verinec XML Schema. Anomaly seems to be targeted at Unix hosts only, while a central concept in the Verinec project is device independence. Work on the Anomaly project seems to have ceased, the Internet site of the project is no longer on-line.

It would be interesting to experiment with the implications of environmental acquisition for modelling network configuration and for simulating networks containing mobile devices. This would enable us to fully implement the ideas of the use case in Section 4.3. In the next section, the concept of environmental acquisition is introduced. After that, the concept is applied to network configuration management and illustrated with an example from network management.

General Concept of Environmental Acquisition

Environmental acquisition is the process of acquiring information from the current container of an object [Gil and Lorenz, 1996]. While Verinec would use the concept within the context of network configuration and simulation, the concept itself is more general. It can be applied to all domains having objects and containment hierarchies.

Object-oriented systems use the notion of "class" to form hierarchies of types. A subclass *is a* kind of its superclass. It will thus inherit all features from the superclass, possibly extending them. The features are all known at compile-time and do not change during execution. Instances of classes have the features of their own class as well as their superclasses.

The type hierarchy is, however, not the only important relation an object can have. Some features depend on the containment structure. An example may illustrate this. A manufacturer of a mobile phone offers different colour variants of the same model. The device consists of electronics, a display and a casing. The display, microchips and casting mould will not change simply because a different colour is used for the casing. What is of interest is only the finish. The casing can be further divided into the front, the back and the keys. Each key is of the same shape and size.

The schematic containment hierarchy is shown in Figure 10.2. The parts of the casing will have a common base class, showing the material used and similar information. For one variant of the device, the keys should have the same colour as the casing front. If inheritance were used to ensure this, we would have to pretend that each key *is a kind of* casing front. Clearly, considering

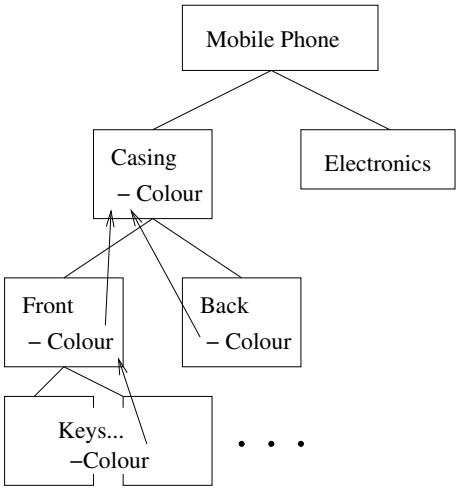


Figure 10.2: Containment hierarchy of a mobile phone.

the front as a *container* for the keys and letting the keys acquire their colour from the container is more natural.

Acquisition allows for environmental polymorphism. Objects of the same class can show different variants of behaviour depending on their environments. This is not to be confused with subtype polymorphism, where code written to operate for a class is also able to handle all of its subtypes. In subtype polymorphism, the code is parametrised by the actual object it operates on, while in environmental acquisition, an object is parametrised by its container.

In opposition to subclassing, a contained object should not acquire all of the features of its container. For a proper design, [Gil and Lorenz, 1996] propose to explicitly list the features that are acquired from the container. To avoid long lists of features, it is possible to form logical groups and simply declare the whole group to be acquired. In the mobile phone example, this could mean that the keys acquire material surface properties like colour or roughness from the body. The shape of each key, or information about the quantity of raw materials needed to build it, on the other hand, is a type of information that should not be acquired but inherited.

Acquisition is implicitly used in programming concepts, for example, in many of the design patterns [Gamma et al., 1995], or within the Model-View-Control (MVC) method.

Acquisition in Network Configuration

The role model defines configuration fragments that encapsulate generic configuration rules. The fragments can be parameterised when referenced in a role. Roles form a class hierarchy, each node *is a* certain type of machine. However, some information is unrelated to the role of a node, but relates to its environment. For nodes in the same subnet, many features are the same, some even must be the same. The most obvious is the subnet mask, which also tells which IP addresses are valid. Examples on the services level are the file server and the printer a client machine has to use.

One can create special roles for each subnetwork. However, if the workstations in subnet A do not differ from other workstations except in some minor parameters, the roles are rather artificial. The limitations of such an unclean design become clear when fundamentally different roles are used in the same subnet. If a workgroup server and client machines share a common subnet, they need to use the same gateway for accessing the rest of the network. But this does not mean they should be in the same role. One approach could be to allow multiple inheritance, permitting a node to be in the role "client", as well as in the role "in-subnet A". However, subnet membership forms a typical containment hierarchy and is not well represented by an *is a* relationship.

This is where environmental acquisition comes in. The hierarchical network structure available in Verinec can be used to define common properties for all of the nodes within an area. For example, it is possible to choose the correct default printer based on the physical location in a certain office. The network mask and default gateway are defined by the subnet a network element is in. The variables discussed in Section 8.2.3 are resolved in lexical scope, permitting a weak form

of environmental acquisition. Usually, however,information is referenced that is already available as part of some configuration. Putting this information into a variable is awkward.

Environmental acquisition could be expressed using XPath[3]. The XPath language allows one to reference XML fragments relative to the current location in an XML document. An example of XPath is the expression '`ancestor::node/@hostname`'. It evaluates to the value of the hostname attribute of the node element containing this XPath. Instead of using an absolute value, the node configuration acquires the information from the environment. What is more, the configuration fragments in the role definition can also make use of the environment. A node can be completely configured by specifying its name, its role and the context it is in.

What is important in environmental acquisition is that XPath references are resolved after variable expansion and target resolution have taken place. If an XPath is used within a global target in `tr:typedef`, it should be resolved in the context of every place where the target is used. An environmental acquisition implementation should use XPath expressions like variables. The curly brackets notation introduced for variables in Section 8.2.3 could be used. After an exclamation point, the parser would expect an XPath it has to evaluate in the current context: `{!XPath}`.

XPath expressions are sufficient for acquiring from parent elements. However, when acquiring from the network, some definitions are required. The nodes in Verinec are considered as composites of hardware and the services they offer. The interfaces are bound to exactly one network and can acquire information from it. The services are contained in the node, but not bound to a specific interface. Precedence must be specified, in order to decide from which interface's environment the information will be acquired.

An Example Network

Figure 10.3 shows the different relations of nodes to their contexts and roles. The machines `diufpc01` and `diufpc02` have the 'client' role, which defines many properties, such as routing setup or user accounts. However, the information on the default gateway, default printer and the netmask is taken from the network of office 404, their environment. The server `diuf-serv` is in a completely different role, but acquires the gateway, the default printer and the netmask from the same environment. The `d-gateway`, finally, is not only part of the office 404 network. Its inbound interface can still acquire the netmask, but the rest must be treated differently. The definition of interface precedence for acquisition is important here. Otherwise, the gateway might have the default gateway of the office 404 network assigned to it, which it is itself.

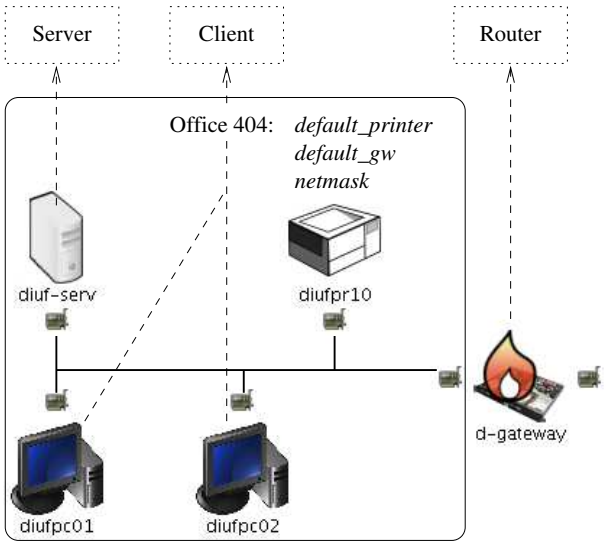


Figure 10.3: Network configuration with environmental acquisition.

The gateway is a good example of why acquisition should never occur implicitly, but should be

[3]For an introduction to the XML Path Language, see [Harold, 2003].

explicitly declared. Let as assume that the acquisition would happen automatically. The gateway would be given the configuration for a default printer, which is nonsense. At the same time, the printer would be assigned a default gateway, which is also not applicable.

XPaths can be used for more than referencing attribute values of their environment. An example is host name to IP address mapping. DHCP and DNS configurations contain about the same information: a mapping of IPs to host names and back. Host names and IP addresses configured statically on the node interfaces are also difficult to keep synchronised with the DNS server. It is redundant to define the name to IP mappings again within the DNS server. However, the DNS server is not contained in the nodes, but needs to acquire the mapping from the nodes. In this case, an object needs information from the objects surrounding it. With XPath evaluation, the environment could define a list of mappings between host names and their static IPs. The DNS server can use the complete list for its database, and every node can select its own IP from the list by using XPath.

Potential of Environmental Acquisition for Network Management

Verifying the configuration (for example, by using the Verinec simulator) is only a last step one resorts to in detecting errors, before the configuration is applied. It is better to avoid introducing mistakes into the configuration in the first place. Environmental acquisition helps realise this goal by avoiding redundant information, thus reducing the potential for conflicting settings.

Redundancy in configuration data is a problem for network setups, as there is a considerable risk of forgetting to update every occurence of a mistake. In larger settings, it is practically impossible to administrate all aspects of each device manually. The network defines a natural containment structure that information can be acquired from. Additional containments that are not directly visible in the network topology, such as rooms or floors, can be taken into account as well. The concept is quite powerful for reducing redundancy, thus ensuring the consistency of configuration data. At the same time, the effort involved in changing the configuration is diminished. Even changing a setting that affects all workstations, such as the name of the file server, need only be done once; all stations are then updated accordingly.

10.2.3 Verinec GUI

Verinec configuration is represented in XML. This format is very suitable for translation and as a base for the simulator; it can also be generated on import. However, editing the configuration in XML is not convenient. At the moment, only the network layout and the interface configuration are supported by a human-friendly GUI. An editor would be needed for each service. Both in adaptation (see Section 8.4.2) and simulation test cases (see Section 8.7.2), XPath expressions are generated to identify offending configuration elements. It would be great if the editor would interpret the XPaths to highlight the corresponding configuration elements.

The abstraction mechanisms of roles and templates should be supported by the editor as well. Role definition and attribution to nodes would be needed. The GUI could be made task-oriented, changing its view according to the node roles. The role model could be extended to a system of network design patterns. This would allow one to do *network engineering* in the same way as software engineering.

The editor should also be improved for handling large networks. A tree structure of the nodes and search functions would be useful for the user to locate nodes. Currently, all of the nodes in the network are loaded when a repository is opened, limiting the possible number of nodes unnecessarily. The logical networks should be editable in the editor and it should be possible to zoom away from individual nodes to network segments. Layouting the logical networks involves complicated decisions, as nodes may belong to several networks.

It would be nice if Verinec could generate printable network documentation. This should be possible using the XML technology eXtensible Stylesheet Language Formatting Objects (XSL-FO). XSL-FO allows one to layout documents based on XML input.

10.2.4 Adaptation

The adaptation module provides a framework for automated configuration. In the current implementation, only a couple of services are defined, and thus only a few servers can be configured. Verinec should support more services and be able to translate each service for all important target systems. To implement new services, a developer currently has to edit the node XML Schema. The Schema concept should be modified to simplify the integration of third-party XML Schemas for service configurations. It should be possible to add Schemas, translators and code for new services and targets, without touching the Verinec program code or core XML Schemas. Each service would have its own namespace. This would also allow one to integrate existing service configuration XML Schemas into Verinec.

Some services that could be implemented are:

Dynamic Routing: For example RIP, OSPF or BGP.

User management: Manage user names and passwords, group memberships, home folders, quota and other user-specific information.

File sharing: Manage mounting and providing remote file systems.

Printer setup: Choose and configure printers to use.

Further services such as Web, SMTP, POP3, IMAP, FTP or SQL databases could be supported. It is, however, questionable, whether the Verinec approach is suited to such complex services. FTP might be simple enough for abstracted configuration. But web- and database servers have very complex and different features. For them, Verinec could support only rudimentary configuration options, or would have to have large restrictions for all implementations. Administrators often need to carefully tune the servers. Abstract configuration would not be able to produce an equally optimised system. On the other hand, one sometimes comes across a heterogeneous web server family, for example, when different applications need to be supported. If the differences between the implementations can be overcome, unified management will probably be highly appreciated.

The network interfaces could also be modelled with more detail to support capacity planning. Currently, there exists simply one network type for Ethernet. The various classes (10MBit, Fast Ethernet, 1000MBit) and media (wire, fibre optics) should be identifiable and other topologies, such as token ring, could be considered. The editor is already prepared for this, defining the network types in an XML document (see Section 8.1.2), but the node Schema would have to be updated.

The current restrictors simply report constructs in the configuration that are omitted by the translator. This is a problem for verification, since the simulator operates on the complete abstract configuration. The restrictors should be extended, so that they can translate Verinec XML back into itself again, removing everything that cannot be translated in the process. This reduced configuration would be used for the simulation, making errors that are due to translation problems detectable. The translator could then expect to find only configuration that can be translated.

For some services, new distribution methods probably have to be found. Support for SNMP has already been developed for the Cisco distributor, but is only used for triggering tftp file transfer. A full SNMP scripting language for the translator output could be specified. The distributor would have to be tested with a suitable target device.

The type system should be supported by the GUI. The module should provide an editor to manipulate node types and to assign types to PCNode elements. It would be nice to also provide a quick graphical overview of which type nodes are and what services they run. The report on translation problems is also very rudimental. Instead of just showing the node XML document and the resulting restriction document, the report should interpret the output and provide nice messages. As suggested in Section 10.2.3, the adaptation module should use the XPath in the report to locate the problematic settings in the configuration.

10.2.5 Import

Both phases of the import module could be improved. Traffic analysis is rather complete, but could work more smoothly. The configuration parsing part is more of a proof-of-concept for now, more parsers are needed.

Traffic Analysis

The traffic analysis part produces only rudimentary data. It would be nice if the Nmap output would be used to build not only `generic-service` elements, but the correct service elements where available. As far as possible, node types for the adaptation should also be created. After the Nmap scan or after parsing the configuration data, the service implementation is known. The XSL translator definitions would probably need to provide some meta data to determine automatically which translator can be used for an implementation. The Nmap names do not necessarily match the translator names.

Currently, the analysis always generates new nodes. It would be interesting to merge scan results into an existing network definition. Problems would arise when there are conflicts. Should Verinec rather place its trust in the configuration it already has in its database? Maybe a human has carefully produced the accurate information and the scan produces faulty results. Or should it trust the newly gathered information about the current network? It is more recent then the information existing in the Verinec database, this database might be outdated.

Parsing of Configuration Data

We implemented only two parsers, one for Netfilter and one for the network interfaces of Red Hat Linux. Other services and devices should be supported. For Linux, scp and remote execute are probably sufficient to access the configuration. However, parsers for each format have to be implemented. Windows can probably be covered by WMI for services supporting this system and some more configuration file parsers. SNMP could be used for importing configuration of suitable devices, since read only information is all the importers need. For Cisco, the complete procedure of transferring the running-config or startup-config to the Verinec machine has already been implemented. A parser for the Cisco IOS configuration format is all that is missing.

A major problem with integrated management solutions is when configuration was manually changed on a device. To quickly solve an urgent problem, an administrator might change the configuration of a firewall directly on the console. Such fixes are not likely to be updated in the NMS by the administrator later on, when the urgency has passed. If the configuration on a device is more recent than the one of Verinec, the adaptation module will just overwrite the corrected configuration with the faulty configuration from the Verinec database. If the import is precise enough, it could be used to retrieve the current configuration and to compare it to the configuration that was last distributed. If any discrepancies were found, the administrator could be warned.

Using configuration parsers, Verinec can import detailed service configuration. This approach could be extended, allowing users to move their entire system configuration from one system to another. For example, Linux distributions often have different configuration files layouts or different syntax, but the functionality is the same. It would be nice to be able to switch between distributions without having to reconfigure every service each time. Verinec can parse the configuration of the old system, generate abstracted Verinec XML from it, and then translate and distribute it to the new system.

10.2.6 Verification

The verification module currently supports only a few services in the application layer. When additional services can be configured in Verinec, the simulator should be extended to support them as well. Section 10.2.4 lists a number of possible services that could be supported.

It would also be interesting to mix simulator sockets and real network sockets. Routing traffic into and out of the simulation opens up some rewarding possibilities. Non-Java applications could be integrated, allowing the use of services not implemented in Verinec, or tracking down problems related to a specific implementation. Moreover, network appliances could be integrated. This would not only be interesting for testing network configuration, but also for debugging devices or server applications in a controlled environment.

Test Cases for Network Configuration

Section 8.7.2 suggests using the simulator on a network and checking the outcome with CLiXML. The best solution would be to completely automate the tests. A simulation test describes two sets of information:

- Input events for the simulator
- CLiXML tests to validate the simulation output.

In this case, the CLiXML tests would not be run against the configuration, but against the XML log file resulting from the simulation. A test can be used to anticipate both *success of a desired behaviour* as well as *failure of an undesired behaviour*. A desired behaviour could be, for example, "nodes inside the firewall must be able to access the Intranet web server". This is tested with the input event of a connection attempt by a node in the intranet, and with a CLiXML test that checks for a HTML success response. Nodes outside the firewall must not be able to contact the Intranet server. A similar connection attempt, originating from a node outside the firewall, should result in either a connection refused message, or in a timeout of the request in the simulation log.

The test automation is currently not fully implemented in Verinec. Simulation results (the event log) have to be saved to a file and CLiXML has to be run with the test against the saved file. However, it should not be too difficult to implement test cases. A simulation test case could look like Listing 39.

Listing 39: SimulationTestCase.xml

```
<TestCase xmlns="http://diuf.unifr.../verification"
          xmlns:clix="http://www.clixml.org/clix/1.0"
          xmlns:ev="http://diuf.unifr.../events">
 <request>
  <ev:event time="0" node="server2" layer="5" service="application"
            id="input23234">
   <ev:application type="launch" program="wget"
                   parameters="http://intranet.network.org" />
  </ev:event>
 </request>
 <expect>
  <clix:exist var="success" in="//ev:event[@id='input23234']/\
                               descendant::ev:event[@node='server2']">
   <clix:equal op1="$success/ev:application/@type" op2="'success'"/>
  </clix:exist>
 </expect>
</TestCase>
```

Environmental Acquisition and Simulation of Mobile Networks

Simulation is of central importance in testing configuration with Verinec. The concept of environmental acquisition introduced in Section 10.2.2 can be used for simulation as well. It can not only be used for defining redundancy-free configuration, but also for simulating and testing the configuration of a node in a dynamic environment.

Working with the concept of mobility produces the added difficulty that the environment changes during the simulation. A mobile device with WLAN capabilities can change its position within the network and thus move into or out of the range of base stations. For statically wired networks, the network definition tells the simulator at startup which interfaces a packet sent over a cable will arrive at. Unlike wired networks, the wireless environment can determine only during runtime which packets will be received by which nodes. The mobile device needs to be within range of a base station in order to communicate. When a device or base station sends a packet, the simulation framework needs to know which devices are in range, so that it can decide where to schedule the packet's reception.

Wireless network client drivers can often be configured to connect automatically to a network. Different aspects, such as encryption and signal quality, can be taken into account. However, since Verinec has no notion of signal quality, automatic reconnection cannot be simulated. Signal quality is neglected for several reasons: the aim of Verinec is to decide whether the configuration is semantically correct. Signal quality is quite hard to predict, and simulating it would make the framework much more complicated. A simplified concept could be used: special events explicitly tell the simulator that a device has moved into or out of the range of a base station, as well as when it should initiate a connection to a network. The necessary events for this concept are *get in range of*, *get out of range of*, *connect* and *disconnect*. Connect tells the higher level services to simulate DHCP or similar protocols to establish a connection with the new network, and requires being in range of the specified base station. Listing 40 shows an extract from an event input file where a mobile device first loses its connection to the base station "hall", then comes back into the range of the base station "room" and connects to it.

Listing 40: Input events to change the environment

```
<event time="0" node="wdevice1">
  <wlan-range type="out" base-station="hall" />
</event>
<event time="1" node="wdevice1">
  <wlan-range type="in" base-station="room" />
</event>
<event time="3" node="wdevice1">
  <wlan-connect base-station="room" />
</event>
```

From the device's point of view, changing its position means moving into a new environment. For the other devices, such as mobile devices in an ad hoc network, or for the base station in an infrastructure mode network, the mobile device that is moving around changes their environment. Whenever a packet is sent over a wireless network interface, the environment of the interface is consulted in order to assess which interfaces are currently in range and able to receive the packet.

This allows one to test the configuration of the services involved in handling mobile devices. As an example, for employees connecting their notebooks from outside the company network using a Virtual Private Network (VPN), the security policy might stipulate that they cannot access a file server with confidential files. However, if the same employee takes her notebook to the office, she should be able to use all of the services. This example also shows that, in configuration tests, mobile does not automatically imply wireless. The concept can also be used with devices that connect by cable to different networks over the course of time.

Mobility does not only have an impact on the simulation framework, but also on the mobile device's run-time configuration. When changing from one base station to another, the mobile device might also change its IP address and default gateway, which can either be static or received dynamically via DHCP. Once the mobile device has an IP address, its behaviour in the new environment can be tested using the appropriate test cases with input events and expected outcomes. However, this dynamic configuration has to be achieved by simulating network protocols. A realistic simulation is crucial to realising the goal of having configuration that will behave correctly in a real network. Environmental acquisition cannot be used directly to update the mobile device's configuration, as it will not be available later on in the real world.

Environmental acquisition, as proposed here, helps to model mobile devices that move in and out of reach of various wireless stations. Because the focus is on the correctness of configuration and not on the details of wireless protocols, acquisition enables one to simulate exactly the required level of detail.

10.2.7 Versioning Configuration

The current Verinec concept does not take versioning into account. The configuration repository holds only the current version of configuration and does not include any history or changelog. Tracking changes is important for several reasons:

Trackability: It is important to know who changed a configuration and when. A changelog helps one to understand why changes were done. This is necessary when several people work on the same network - but it is useful even for a single administrator, as it is impossible to remember the reasons for all of the configuration modifications that are done. If something goes wrong, one can determine who caused the problem and how long it has existed.

Backups: Having older versions of the configuration allows one to switch back to a setup that is known to work, in case the last update has destroyed something. Verinec, or even a perfect configuration tool, cannot always prevent errors from happening, such as problems due to bugs in implementations.

Compare: Concerning distribution, it would be useful to know what configuration was last distributed to a device. This would enable one to detect modifications done directly on the device so that the user can be informed of potential problems. If devices are constantly monitored, it becomes impossible to edit configuration out of sync with the device.

10.2.8 Access Control

As soon as more than one administrator works with Verinec, access control is needed. Access control for the configuration settings is important in large installations, as several administrators have to work together to maintain the network. Some may be trusted with full access to all settings, but some should only be able to modify the settings in the area they are responsible for. Authentication is definitely required for tracking who made changes to the configuration (see Section above). Access control could specify permissions on a fine-grained level, based on configuration elements.

Sub network: IP range, domain, environment

Role of computer: Server, student PC, router

Service: Network layer or specific service such as DHCP or user management

An interesting project concerned with access control for administrators is 'ona' [Campbell, 2005]. This web-based configuration system for switches provides a granular access control for administrators and groups. Every single port can be part of a group, allowing permissions to be controlled on a lower level than would be possible on the device itself.

The combination of subnets (see Section 8.2) and the role hierarchy proposed in Section 10.2.1 makes a fine-grained access control possible. Permissions can be specified, based on the role of a node or the subnet a node is in. Write permissions can be restricted to some of the features of a particular service. Combining these properties makes it possible, for example, for an assistant administrator to only add or remove the workstations in one computer pool, but not to change anything other than the names and MAC addresses of those computers. The administrator responsible for DHCP could be permitted to edit all DHCP configuration without having the right to change anything else on the DHCP server machine. Since the simulation knows about the whole network, it will still be able to point out problems, even if an administrator can only modify a limited number of settings and nodes.

However, some additional considerations are needed. When simulating the whole network for somebody with restricted access rights, information about problematical configuration needs to be limited. The error message may not disclose confidential information such as user accounts or other sensitive data to persons who are not qualified to see them. To fix a problem, an administrator with limited rights might need to ask another administrator to change configuration. It would be more efficient if Verinec would support an information flow that asks other administrators to do configuration modifications in their areas of responsibility. This would both solve the problem of information disclosure and simplify locating the problem for the other administrator.

10.2.9 Scalability and Autonomic Systems

Verinec follows a centralised management paradigm. It is focused on specifying the intention of the network configuration and on verifying correctness. It would be impossible to verify all of the

configuration in a distributed concept, since a centralised knowledge of the total configuration is essential for the simulator.

By supporting a framework for management by delegation (see Section 2.4.2), Verinec may be used to specify the goals for intelligent agents. The implementation that informs the agents of their goals could be straightforward, if it were done by creating appropriate distribution modules. A solution would have to be found to ensure correctness. Should Verinec compare the various goals for the agents? Or should it simulate the agents and test whether they are able to configure the network? The latter would position Verinec as an agent development tool, rather than as a tool for network configuration.

Verinec could be a good basis for research on self-organising, autonomic systems as proposed by [Kephart and Chess, 2003]. Autonomic systems need "autonomic elements with closed control loop" [Herrmann et al., 2005]. Verinec would then no longer use a central repository containing detailed configuration. An autonomic system is configured on a higher level than the configuration editors Verinec currently provides. Verinec could be used to define the requirements of the network and to automatically find a way of meeting them. It could then create the appropriate configuration or tell the autonomic elements about the goals, depending on how evolved the autonomic system is.

Part IV

Appendix

Appendix A

Acronyms

NMS Network Management System

GUI Graphical User Interface

ISO International Organization for Standardization

OSI Open Systems Interconnection Reference Model: a layered description for network protocol analysis.

IETF Internet Engineering Task Force: an open organisation to decide on common standards for the Internet.

RFC Request For Comment: standards formulated by the IETF are humbly named "request for comments".

LAN Local Area Network: a network with cable connections. Usually Ethernet IEEE 802.3.

WLAN Wireless Local Area Network: network protocol for radio communication defined in IEEE 802.11.

SSID System Set Identifier: name for a WLAN broadcasted by an access point.

WEP Wired Equivalent Privacy: first but unsuccessful attempt to provide security to WLANs. IEEE 802.11b

WPA Wi-Fi Protected Access: up to date secure encryption for wireless networks. Should have been part of IEEE 802.11i, but was published earlier to replace WEP.

MAC Medium Access Control: method used by Ethernet to address packets on OSI layer 2.

IP Internet Protocol: basic protocol on network layer. RFC 791. The term is also used as a short form of *IP address*, the identification numbers for interface participating in the Internet Protocol.

ARP Address Resolution Protocol: protocol for Ethernet to resolve IP addresses to MAC addresses. RFC 826.

TCP Transmission Control Protocol: basic connection oriented protocol on transport layer. RFC 793.

UDP User Datagram Protocol: basic connectionless protocol on transport layer. RFC 768.

AS Autonomous System: a connected group of one or more IP address prefixes run by one or more network operators which has a single and clearly defined routing policy. RFC 1930.

IGP Interior Gateway Protocol: protocols used for routers inside one AS.

EGP Exterior Gateway Protocol: protocols used to route between separate AS.

BGP Border Gateway Protocol: the EGP routing information exchange protocol currently used for the Internet. Current version is BGPv4, RFC 4271.

RIP Routing Information Protocol: an IGP protocol commonly used in the earlier days of networking. RFC 1058.

OSPF Open Shortest Path First: a widely used IGP protocol. Current version is OSPFv3, RFC 2740.

MPLS Multiprotocol Label Switching: emulates a circuit switching network over a packet network. RFC 3031.

ECMP Equal-cost multi-path routing: a strategy for routing over several paths. See also RFC 2991 and 2992.

DNS Domain Name System: translates the alphabetic domain names into IP addresses. RFC 1034/1035.

SSL Secure Socket Layer: an implementation of network sockets providing encrypted data transfers. Also called TLS, RFC 2246.

ssh secure shell: a protocol to log into remote machines and get a command shell. RFC 4250–4254.

FTP File Transfer Protocol: unencrypted protocol to exchange files. RFC 959.

tftp Trivial FTP: simplyfied version of FTP, without authentication and server full filename has to be known in advance. RFC 1350.

DHCP Dynamic Host Configuration Protocol: allows to use a central server to assign IP address, host name and other properties to hosts dynamically. RFC 2131.

SNMP Simple Network Management Protocol: industry standard protocol for network management. RFC 1157.

TTL Time To Live: packet counter decreased at each step. When 0, the packet is discarded.

NAT Network Address Translation: to hide internal network details, gateways sometimes modify packets to bear their address before forwarding them. They have to track the connections and translate the IP address back when receiving responses from the target host.

VPN Virtual Private Network: provides a encrypted connection to a trusted network over an insecure network, typically the Internet. Applications can send data like usual in a local network, the VPN is not visible for them.

DMZ Demilitarised Zone: a security concept to protect servers from the Internet and the local network from the servers.

MTBF Mean Time Between Failure: the time after which a device had a failure with probability of 50%.

IOS Internet Operating System: operating system for Cisco devices.

XML Extensible Markup Language [Quin et al., 2003].

DTD Document Type Definition: the first syntax to define the valid structure of an XML document. W3C.

CLiXML Constraint Language in XML: a language to formulate semantic requirements on an XML document.

XSL eXtensible Stylesheet Language: used to translate XML documents. There are two different parts: Transformations for changing the format (XSLT) and Formatting Objects (XSL-FO) to layout XML data.

XSLT eXtensible Stylesheet Language Transformations: transforms XML documents into other XML or plain-text documents [Clark, 1999].

XSL-FO eXtensible Stylesheet Language Formatting Objects: renders XML documents to a graphical representation. W3C.

DMTF Distributed Management Task Force: a non-profit organisation with the goal of developping common standards for management.

WBEM Web-based enterprise management: collection of protocols for management defined by the DMTF, basing on standards like XML and HTTP.

CIM Common Information Model: core of the WBEM protocol suite.

CIMOM CIM Object Manager: concerned with creating CIM objects.

WMI Windows Management Instrumentation: Microsoft standard for automated system management, based on the Distributed Component Object Model (DCOM).

WQL WMI Query Language: a subset of SQL for selecting WMI objects.

SQL Structured Query Language: a declarative language to manipulate relational databases.

JNI Java Native Interface: to use external shared libraries inside Java.

ITIL IT Infrastructure Library: a series of books with best practices for IT management.

SOX Sarbanes-Oxley Act: law in the united states about data protection, including network security.

PCI Payment Card Industry Data Security Standard

Bibliography

[Abrahamson et al., 2003] Abrahamson, C., Blodgett, M., Kunen, A., Mueller, N., and Parter, D. (2003). Splat: A network switch/port configuration management tool. In *Seventeenth Systems Administration Conference (LISA 03)*, Berkeley, CA, USA. Usenix. Available from: `http://www.usenix.org/event/lisa03/tech/full_papers/abrahamson/`.

[Aebischer, 2004] Aebischer, P. (2004). Network Sniffer, ein Modul für Verinec. Bachelor's thesis, University of Fribourg, Switzerland. Available from: `http://diuf.unifr.ch/tns/projects/verinec/students/SnifferPatrickAebischer.pdf`.

[Agilent, 2006] Agilent (2006). *Network Analyzer product group*. Available from: `http://www.agilent.com/`.

[Aitchison, 2007] Aitchison, R. (2001–2007). *DNS for Rocket Scientists*. Zytrax, Inc, Montreal. Available from: `http://www.zytrax.com/books/dns/`.

[Anderson and Burgess, 2001] Anderson, E. and Burgess, M. (2001). *Selected Papers in Network & System Administration*. John Wiley & Sons, Inc., New York; London.

[Anderson and Scobie, 2002] Anderson, P. and Scobie, A. (2002). LCFG - the Next Generation. In *UKUUG Winter Conference*. UKUUG. Available from: `http://www.lcfg.org/doc/ukuug2002.pdf`.

[angryziber, 2006] angryziber (2006). *Angry IP Scanner*. Available from: `http://www.angryziber.com/ipscan/`.

[Antener, 2005] Antener, G. (2005). Configimporter. Master's thesis, University of Fribourg, Switzerland. Available from: `http://diuf.unifr.ch/tns/projects/verinec/students/ConfigImporterGeraldineAntener.pdf`.

[Aprisma, 2006] Aprisma (2006). *SPECTRUM Configuration Manager*. Available from: `http://www.aprisma.com/products/ecm.shtml`.

[Awesso, 2005] Awesso, R. (2005). DJBDNS and DHCP translators. Bachelor's thesis, University of Fribourg, Switzerland. Available from: `http://diuf.unifr.ch/tns/projects/verinec/students/DJBDNSRobertAwesso.pdf`.

[Barringer et al., 2004] Barringer, H., Goldberg, A., Havelund, K., and Sen, K. (2004). Rule-based runtime verification. In *Proceedings of Fifth International VMCAI conference (VMCAI'04)*. Springer.

[Bellogini and Santarelli, 2004] Bellogini, A. and Santarelli, I. (2004). *Network Markup Language*. Available from: `http://giga.dia.uniroma3.it/~ivan/NetML/`.

[Bernstein, 2007] Bernstein, D. J. (2007). *djbdns: Domain Name System tools*. Available from: `http://cr.yp.to/djbdns.html`.

[Bhargavan et al., 2002] Bhargavan, K., Gunter, C., Kim, M., Lee, I., Obradovic, D., Sokolsky, O., and Viswanathan, M. (2002). Verisim: Formal Analysis of Network Simulations. *IEEE Transactions on Software Engineering*, 28(2):129–145.

[Blouin et al., 2003] Blouin, F. J., Sack, A., Grover, W. D., and Nasrallah, H. (2003). Benefits of p-Cycles in a Mixed Protection and Restoration Approach. In *Design of Reliable Com. Networks (DRCN03)*, pages 203–211, Alberta, Canada. University of Alberta.

[Buchmann et al., 2005] Buchmann, D., Jungo, D., and Ultes-Nitsche, U. (2005). Automated Configuration Distribution in Verinec. In *Proceedings of the 2nd International Conference on e-Business and Telecommunications Networks (ICETE 2005)*, Reading, UK. INSTICC Press.

[Buchmann et al., 2006] Buchmann, D., Jungo, D., and Ultes-Nitsche, U. (2006). Environmental acquisition in mobile network simulation. In *Proceedings of the International Conference on Wireless Information Networks and Systems (WINSYS 2006)*, Setubal, Portugal. INSTICC Press.

[Buchmann et al., 2007a] Buchmann, D., Jungo, D., and Ultes-Nitsche, U. (2007a). Improving network reliability by avoiding misconfiguration. In *Proceedings of the 6th International Workshop on the Design of Reliable Communication Networks (DRCN 2007)*, La Rochelle, France. Accepted for publication, to appear in October 2007.

[Buchmann et al., 2007b] Buchmann, D., Jungo, D., and Ultes-Nitsche, U. (2007b). A role model to cope with the complexity of network configuration. In *Proceedings of the 3rd International Network Optimization Conference (INOC 2007)*, Spa, Belgium. Université Libre de Bruxelles.

[Burgess, 1995] Burgess, M. (1995). Cfengine: a site configuration engine. In *USENIX Computing systems*, volume 8.3, University of Oslo, Norway. Usenix.

[Burgess, 1998] Burgess, M. (1998). Computer Immunology. In *Proceedings of the 12th System Administration Conference (LISA 1998)*, Berkeley, CA, USA. Usenix. Available from: `http://www.usenix.org/publications/library/proceedings/lisa98/burgess.html`.

[Caldwell et al., 2003] Caldwell, D., Gilbert, A., Gottlieb, J., Greenberg, A., Hjalmtysson, G., and Rexford, J. (2003). The cutting EDGE of IP router configuration. In *Second Workshop on Hot Topics in Networks*. Networks and Mobile Systems Group, MIT. Available from: `http://nms.lcs.mit.edu/HotNets-II/papers/ip_router.pdf`.

[Campbell, 2005] Campbell, B. (2005). Open Network Administrator (ONA). In *Nineteenth Systems Administration Conference (LISA 05)*, Berkeley, CA, USA. Usenix. Available from: `http://www.usenix.org/event/lisa05/tech/full_papers/campbell/`.

[Cancela and Khadiri, 1996] Cancela, H. and Khadiri, M. E. (1996). A Simulation Algorithm for Source Terminal Communication Network Reliability. In *29th Annual Simulation Symposium (SS '96)*. IEEE Computer Society.

[Castelli, 2001] Castelli, M. J. (2001). *Network Consultants Handbook*. Cisco Press.

[Chang et al., 2004] Chang, Y.-R., Amari, S. V., and Kuo, S.-Y. (January 2004). Computing System Failure Frequencies and Reliability Importance Measures Using OBDD. In *Transactions on Computers*. IEEE Computer Society.

[Check Point Inc., 2007] Check Point Inc. NGX Platform [online]. (2007). Available from: `http://www.checkpoint.com`.

[Chen and Yuang, 1995] Chen, Y. G. and Yuang, M. C. (1995). An efficient terminal-pair reliability algorithm for network management. In *20th Annual IEEE International Conference on Local Computer Networks (LCN'95)*, Washington, USA. IEEE Computer Society.

[Cheswick et al., 2003] Cheswick, W. R., Bellovin, S. M., and Rubin, A. D. (2003). *Firewalls and Internet Security*. Addison Wesley Professional, Boston, 2nd edition.

[Chudley and Ultes-Nitsche, 2002] Chudley, S. R. and Ultes-Nitsche, U. (2002). Simulation and Implementation of an E-Commerce Communications Infrastructure using XML Specifications. In *Proceeding of Business Information Systems (BIS) 2002*, Poznan, Poland. Poznan University of Economics.

[Cisco Systems, 2003] Cisco Systems (2003). *Cisco 826, 827, 828, 831, 836, and 837 and SOHO 76, 77, 78, 91, 96, and 97 Routers Software Configuration Guide*. Cisco Press. Available from: `http://www.cisco.com/application/pdf/en/us/guest/products/ps380/c2001/ccmigration_09186a008015be23.pdf`.

[Cisco Systems, 2005] Cisco Systems (2005). *Cisco Systems Corporate Timeline*. Available from: `http://newsroom.cisco.com/dlls/corporate_timeline.pdf`.

[Cisco Systems, 2007a] Cisco Systems. Configuring the Catalyst Switched Port Analyzer (SPAN) Feature [online]. (2007). Available from: `http://www.cisco.com/warp/public/473/41.html`.

[Cisco Systems, 2007b] Cisco Systems. Internet Operation System [online]. (2007). Available from: `http://www.cisco.com`.

[Clark, 1999] Clark, J. (1999). XSL Transformations (XSLT). Technical report, W3C. Available from: `http://www.w3.org/TR/xslt`.

[Combs et al., 2006] Combs, G. et al. (2006). *Wireshark: A Network Protocol Analyzer [Formerly Ethereal]*. Available from: `http://www.wireshark.org/`.

[CommView, 2006] CommView (2006). *CommView - Network Monitor and Analyzer*. Available from: `http://www.tamos.com/products/commview/`.

[Cowie et al., 1999] Cowie, J., Nicol, D. M., and Ogielski, A. T. (1999). Modeling the global internet. *Computing in Science & Engineering*, 1(1):42–50.

[Crispin and House, 2002] Crispin, L. and House, T. (2002). *Testing Extreme Programming*. Addison-Wesley, Reading, MA.

[Dijkstra, 1959] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271.

[DMTF, 2007] DMTF. Distributed Management Task Force, Inc. [online]. (1996-2007). Available from: `http://www.dmtf.org/`.

[DMTF, 2004] DMTF (2000-2004). Specification for CIM Operations over HTTP. Technical report. Available from: `http://www.dmtf.org/standards/published_documents/DSP200.pdf`.

[Dooley, 2002] Dooley, K. (2002). *Designing large-scale LANs*. O'Reilly, Sebastopol, 1st edition.

[Ehret, 2005] Ehret, C. (2005). Cisco translator. Master's thesis, University of Fribourg, Switzerland. Available from: `http://diuf.unifr.ch/tns/projects/verinec/students/CiscoChristophEhret.pdf`.

[Fock and Katz, 2006] Fock, F. and Katz, J. Snmp4j - the object oriented snmp api for java managers and agents [online]. (2003–2006). Available from: `http://www.snmp4j.org`.

[Foster, 2002] Foster, I. (2002). What is the Grid? A Three Point Checklist. Available from: `http://www.gridtoday.com/02/0722/100136.html`.

[Foster and Kesselmann, 1998] Foster, I. and Kesselmann, C. (1998). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, San Francisco, California, 1st edition.

[Fujii, 2007] Fujii, K. (2000–2007). *Jpcap - Java package for packet capture*. Available from: `http://netresearch.ics.uci.edu/kfujii/jpcap/doc/`.

[Fyodor, 1997] Fyodor (1997). The art of port scanning. *Phrack Magazine*, 7(51). Available from: `http://www.phrack.org/show.php?p=51&a=11`.

[Fyodor, 2006] Fyodor (1997–2006). *Network Mapper*. Available from: `http://www.insecure.org/nmap/`.

[Fyodor, 1998] Fyodor (1998). Remote os detection via tcp/ip stack fingerprinting. Technical report, Insecure.org. Available from: http://www.insecure.org/nmap/nmap-fingerprinting-article.txt.

[Gamma et al., 1995] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley.

[Garschhammer and Schiffers, 2005] Garschhammer, M. and Schiffers, M. (2005). Integrated IT-Management in Large-Scale, Dynamic and Multi-Organizational Environments. In *Proceedings of the 12th Annual Workshop of HP OpenView University Association*. Hewlett Packard. Available from: http://www.mnm-team.org/pub/Publikationen/gasc05/PDF-Version/gasc05.pdf.

[Gertner, 2005] Gertner, N. (2005). Pci compliance: Don't become another headline. *ZDNet News*. Available from: http://news.zdnet.com/2100-1009_22-5823634.html.

[Gil and Lorenz, 1996] Gil, J. and Lorenz, D. H. (1996). Environmental Acquisition – A New Inheritance-Like Abstraction Mechanism. In *Proceedings of the 11th Annual Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 214–231. OOPSLA'96, Acm SIGPLAN Notices. Available from: http://www.ccs.neu.edu/home/lorenz/papers/oopsla96/.

[Goldszmidt and Yemini, 1995] Goldszmidt, G. and Yemini, Y. (1995). Distributed management by delegation. In *15th International Conference on Distributed Computing Systems*. IEEE Computer Society. Available from: http://ieeexplore.ieee.org/iel5/3728/10892/00500036.pdf.

[Haldimann, 2005] Haldimann, N. IniEditor - Java library to read and edit INI-style configuration files [online]. (2005). Available from: http://ubique.ch/code/inieditor/.

[Halloway and Gehtland, 2005] Halloway, S. and Gehtland, J. (2000–2005). *Jawin - a Java/Win32 interoperability project*. Available from: http://jawinproject.sourceforge.net/.

[Harold, 2003] Harold, E. R. (2003). *Processing XML with Java*. Addison-Wesley, Boston.

[Herrmann et al., 2005] Herrmann, K., Mühl, G., , and Geihs, K. (2005). Self-Management: The Solution to Complexity or Just Another Problem? *IEEE Distributed Systems Online*, 6(1). Available from: http://csdl2.computer.org/comp/mags/ds/2005/01/o1001.pdf.

[Herzberg and Raz, 2007] Herzberg, M. and Raz, D. (2007). Optimal Assignment of Pre Cross-connected Trails to Shared-Backup Path-Protection Resources. In Fortz, B., editor, *Proceedings of the International Network Optimization Conference (INOC)*, Bruxelles, Belgium. Université Libre de Bruxelles. Available from: http://www.poms.ucl.ac.be/inoc2007/Papers/author.91/paper/paper.91.pdf.

[Hewlett-Packard, 2007] Hewlett-Packard. HP OpenView [online]. (2007). Available from: http://www.managementsoftware.hp.com.

[Hug, 2006] Hug, J. (2006). Verinec Firewall. Master's thesis, University of Fribourg, Switzerland. Available from: http://diuf.unifr.ch/tns/projects/verinec/students/FirewallJasonHug.pdf.

[IBM, 2007] IBM. IBM Tivoli NetView [online]. (2007). Available from: http://www.ibm.com/software/tivoli/products/netview/.

[Intelliden, 2006] Intelliden (2006). *Intelliden R-Series*. Available from: http://www.intelliden.com.

[ISC, 2007] ISC (1994–2007). *Berkeley Internet Name Domain*. Redwood City, CA. Available from: http://www.isc.org/sw/bind/.

[ISO, 1996] ISO (1996). *Enhanced BNF*. ISO/IEC 14977:1996(E). Available from: http://www.cl.cam.ac.uk/~mgk25/iso-ebnf.html.

[Jacobson et al., 2006] Jacobson, V., Leres, C., and McCanne, S. (2001–2006). *tcpdump/libpcap*. Available from: `http://www.tcpdump.org`.

[JTC1/SC34, 2006] JTC1/SC34, I. Document Schema Definition Languages [online]. (2006). Available from: `http://dsdl.org/`.

[Jungo, 2005] Jungo, D. Open clixml [online]. (2005). Available from: `http://clixml.sourceforge.net/`.

[Jungo, 2008] Jungo, D. (2008). VeriNeC - Secure network configuration through verification. unpublished.

[Jungo et al., 2004] Jungo, D., Buchmann, D., and Ultes-Nitsche, U. (2004). The role of simulation in a network configuration engineering approach. In *ICICT 2004, Multimedia Services and Underlying Network Infrastructure*, Cairo, Egypt. Information Technology Institute.

[Jungo et al., 2005] Jungo, D., Buchmann, D., and Ultes-Nitsche, U. (2005). A Unit Testing Framework for Network Configurations. In *Proceedings of the 3rd International Workshop on Modelling, Simulation, Verification, and Validation of Enterprise Information Systems (MSVVEIS 2005)*, Miami, Florida, USA. INSTICC Press.

[Jungo et al., 2006] Jungo, D., Buchmann, D., and Ultes-Nitsche, U. (2006). Testing of semantic properties in XML documents. In *Proceedings of the 4th International Workshop on Modelling, Simulation, Verification, and Validation of Enterprise Information Systems (MSVVEIS 2006)*, Paphos, Cyprus. INSTICC Press.

[Jungo et al., 2007] Jungo, D., Buchmann, D., and Ultes-Nitsche, U. (2007). Assessment of code quality through classification of unit tests in verinec. In *Proceedings of the 21st IEEE International Conference on Advanced Information Networking and Applications Workshops (AINA 2007)*, Washington, DC, USA. IEEE Computer Society.

[Kephart and Chess, 2003] Kephart, J. O. and Chess, D. M. (2003). The Vision of Autonomic Computing. *Computer*, (36):41–50. Available from: `http://www.research.ibm.com/autonomic/research/papers/AC_Vision_Computer_Jan_2003.pdf`.

[Kuzmik, 2006] Kuzmik, R. Local Managed DNS (Java) [online]. (2006). Available from: `http://rkuzmik.blogspot.com/2006/08/local-managed-dns-java_11.html`.

[Law and McComas, 1996] Law, A. M. and McComas, M. G. (1996). Simulation of communications networks. In *Proceedings of the 28th conference on Winter simulation*, New York, USA. ACM Press.

[Lee and Louis, 2006] Lee, R. E. and Louis, J. C. (2006). *Unicornscan*. Available from: `http://www.unicornscan.org/`.

[Leinen, 2006] Leinen, S. Original Van Jacobson/Unix/LBL Traceroute [online]. (2006). Available from: `http://kb.pert.switch.ch/cgi-bin/twiki/view/PERTKB/VanJacobsonTraceroute`.

[Lhotka and Novotny, 2007] Lhotka, L. and Novotny, J. (2002-2007). Netopeer. Technical report, Cesnet. Available from: `http://www.liberouter.org/netopeer/`.

[Liu and Chien, 2004] Liu, X. and Chien, A. A. (2004). Realistic Large-Scale Online Network Simulation. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, Washington, USA. IEEE Computer Society.

[Loeffel, 2004] Loeffel, R. (2004). Verinec Studio, ein GUI für den Verinec Simulator. Bachelor's thesis, University of Fribourg, Switzerland. Available from: `http://diuf.unifr.ch/tns/projects/verinec/students/BaseGuiRenatoLoeffel.pdf`.

[Logan et al., 2002] Logan, M., Felleisen, M., and Blank-Edelman, D. (2002). Environmental Acquisition in Network Management. In *Sixteenth Systems Administration Conference (LISA 02)*, pages 175–184, Berkeley, CA, USA. Usenix. Available from: `http://www.usenix.org/events/lisa02/tech/logan.html`.

[Louden, 1997] Louden, K. C. (1997). *Compiler Construction: Principals and Practice*. PWS Publishing Company, Boston, MA.

[Mahajan et al., 2002] Mahajan, R., Wetherall, D., and Anderson, T. (2002). Understanding BGP Misconfiguration. In *Proceedings of ACM SIGCOMM*. ACM Press. Available from: `http://www.sigcomm.org/sigcomm2002/papers/bgpmisconfig.pdf`.

[Martin-Flatin et al., 1999] Martin-Flatin, J.-P., Znaty, S., and Hubaux, J.-P. (1999). A survey of distributed enterprise network and systems management paradigms. *Journal of Network and Systems Management*, 7(1).

[Matuska, 2004] Matuska, M. (2004). Metaconfiguration of the computer network. In *CESNET technical report 27/2004*, Praha, Czech Republic. Cesnet.

[Mauro and Schmidt, 2001] Mauro, D. R. and Schmidt, K. J. (2001). *Essential SNMP*. O'Reilly, Sebastopol, 1st edition.

[Microsoft, 2003] Microsoft. Wmi: Introduction to windows management instrumentation [online]. (2003). Available from: `http://www.microsoft.com/whdc/system/pnppwr/wmi/WMI-intro.mspx`.

[Milbrandt et al., 2007] Milbrandt, J., Menth, M., and Lehrieder, F. (2007). A Priori Detection of Link Overload due to Network Failures. In *15. Fachtagung Kommunikation in Verteilten Systemen (KiVS 2007)*, Berlin, Germany. Springer.

[Narain et al., 2003] Narain, S., Cheng, T., Coan, B., Kaul, V., Parmeswaran, K., and Stephens, W. (2003). Building autonomic systems via configuration. In *Proceedings of AMS Autonomic Computing Workshop*, Seattle, WA. Available from: `http://www.argreenhouse.com/papers/narain/Autonomic.pdf`.

[ns, 2005] ns (2005). *The Network Simulator - ns2*. Available from: `http://www.isi.edu/nsnam/ns/`.

[Oppenheimer et al., 2003] Oppenheimer, D., Ganapathi, A., and Patterson, D. A. (2003). Why do Internet services fail, and what can be done about it? In *Proceedings of the 4th Usenix Symposium on Internet Technologies and Systems*. USITS '03. Available from: `http://roc.cs.berkeley.edu/papers/usits03.pdf`.

[Ornaghi and Valleri, 2006] Ornaghi, A. and Valleri, M. (2001–2006). *Ettercap NG*. Available from: `http://ettercap.sourceforge.net/`.

[Page et al., 2000] Page, B., Lechler, T., and Claassen, S. (2000). *Objektorientierte Simulation in Java mit dem Framework DESMO-J*. BoD GmbH, Norderstedt.

[Parker, 2005] Parker, J. FCAPS, TMN & ITIL - Three Key Ingredients to Effective IT Management [online]. (2005). Available from: `http://www.openwatersolutions.com/docs/FCAPS_TMN_%20ITIL.pdf`.

[Patterson, 2002] Patterson, D. A. (2002). A Simple Way to Estimate the Cost of Downtime. In *Sixteenth Systems Administration Conference (LISA 02)*, Berkeley, CA, USA. Usenix. Available from: `http://www.usenix.org/events/lisa02/tech/full_papers/patterson/`.

[Performance Technologies, 2001] Performance Technologies, I. (2001). *The Effects of Network Downtime on Profits and Productivity*. White Paper. Available from: `http://www.netcordia.com/news/TheCostOfNetworkDowntime.pdf`.

[Pérez and Bruyère, 2007] Pérez, P. and Bruyère, B. (2007). DESEREC: Dependability and Security by Enhanced Reconfigurability. *European CIIP Newsletter*, 3(1):9–11.

[Quin et al., 2003] Quin, L. et al. (1996-2003). Extensible markup language (xml). Technical report, W3C. Available from: `http://www.w3.org/XML/`.

[RedHat, 2003] RedHat (2003). *Red hat Linux Reference Guide*.

[Sandlund, 2001] Sandlund, K. (2001). Network management using WBEM. Master's thesis, LuleåTekniskа Universitet, Luleå, Sweden. Available from: `http://epubl.luth.se/1402-1617/2001/025/LTU-EX-01025-SE.pdf`.

[Sarbanes and Oxley, 2002] Sarbanes, P. and Oxley, M. G. (2002). Sarbanes-oxley act. In *One Hundred Seventh Congress of the United States of America at the second session*. Available from: `http://www.law.uc.edu/CCL/SOact/soact.pdf`.

[Seifriz, 2006] Seifriz, M. (2006). Traceroute for Network import. Master's thesis, University of Fribourg, Switzerland. Available from: `http://diuf.unifr.ch/tns/projects/verinec/students/TracerouteMartialSeifriz.pdf`.

[Skoudis and Liston, 2006] Skoudis, E. and Liston, T. (2006). *Counter Hack Reloaded*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition.

[Song, 2001] Song, D. (2000–2001). *dSniff - a collection of tools for network auditing and penetration testing*. Available from: `http://naughty.monkey.org/~dugsong/dsniff/`.

[Sweeney et al., 2007] Sweeney, A., Smith, A., Hinson, G., and O'Doherty, N. ISO 17799 and 27001 Wiki [online]. (2007). Available from: `http://iso-17799.safemode.org`.

[Systimax, 2007] Systimax (2007). *Speed and Agility. Performance and Reliability*. Hickory, NC, USA. Available from: `http://www.systimax.com/research/`.

[Tanenbaum, 2003] Tanenbaum, A. S. (2003). *Computer Networks*. Prentice Hall, 4th edition.

[Toiga, 1989] Toiga, J. W. (1989). *Disaster Recovery Planning: Managing Risk and Catastrophe in Information Systems*. Yourdon Press.

[Toledo, 2006] Toledo, J. (2000–2006). *EtherApe*. Available from: `http://etherape.sourceforge.net`.

[Toren, 2006] Toren, M. (2001–2006). tcptraceroute - linux traceroute implementation. Available from: `http://michael.toren.net/code/tcptraceroute/`.

[Vasseur et al., 2004] Vasseur, J.-P., Pickavet, M., and Demeester, P. (2004). *Network Recovery*. Elsevier Science & Technology Books, 1st edition.

[Viega et al., 2000] Viega, J., Bloch, J., Kohno, Y., and McGraw, G. (2000). ITS4: A Static Vulnerability Scanner for C and C++ Code. In *Proceedings of the Annual Computer Security Applications Conference*, Washington, USA. IEEE Computer Society. Available from: `http://www.cse.ucsd.edu/users/tkohno/papers/ACSAC00/`.

[Vogel, 2005] Vogel, D. (2005). Gui node editor. Bachelor's thesis, University of Fribourg, Switzerland. Available from: `http://diuf.unifr.ch/tns/projects/verinec/students/GuiDamianVogel.pdf`.

[Voyence, 2006] Voyence (2006). *Voyence Control NG*. Available from: `http://www.voyence.com`.

[Welte et al., 2006] Welte, H., Kadlecsik, J., Josefsson, M., McHardy, P., and Kozakai, Y. (2006). *The netfilter.org project*. Available from: `http://www.netfilter.org/`.

[Witek, 2005] Witek, L. (2005). tracetcp - windows traceroute implementation. Available from: `http://tracetcp.sourceforge.net`.

[Yeh et al., 2002] Yeh, F.-M., Lin, H.-Y., and Kuo, S.-Y. (2002). Analyzing network reliability with imperfect nodes using obdd. In *Pacific Rim International Symposium on Dependable Computing (PRDC'02)*. IEEE Computer Society.

[Zurkinden, 2005] Zurkinden, N. (2005). Windows adapter. Master's thesis, University of Fribourg, Switzerland. Available from: `http://diuf.unifr.ch/tns/projects/verinec/students/WindowsNadineZurkinden.pdf`.

Curriculum Vitae

David Buchmann
Mittelstrasse 68
3012 Bern
http://www.budda.ch
budda@budda.ch

Geboren am 28. September 1978 in Basel
Heimatort: St. Gallen

Eltern:
Dr. phil. Rudolf Buchmann, Psychotherapeut SPV und
Catherine Buchmann, Psyhcotherapeutin SPV

Ausbildung

2004-2008	Dissertation in Informatik an der Universität Freiburg (CH). Betreuung von Master- und Bachelor-Arbeiten. Abschluss als *Doctor scientiarum informaticarum*.
1998-2003	Zweisprachiges Studium in Informatik mit Nebenfach Medienwissenschaften, Universität Freiburg (CH). Abschluss als *Master of Science* in Informatik.
1993-1998	Mathematisch-Naturwissenschaftliches Gymnasium, Kantonsschule St.Gallen. Abschluss mit Matura Typus C.
1991-1993	Sekundarschule Blumenau, St.Gallen.
1989-1991	Primarschule Rotmonten, St.Gallen.
1985-1989	Primarschule Bruderholz, Basel.

Berufserfahrung und Praktika

ab 2007	Projektleiter und Software Entwickler bei der Firma Liip AG.
2003	Zivildienst beim Service Civil International SCI. Leiten von Lagern mit internationalen Freiwilligen, Mitarbeit bei der Organisation einer Kulturwoche.
2001-2003	Masterarbeit und Teilzeitstelle bei der ESPRiT Unternehmensberatung GmbH. Implementierung des XML-Protokolls SyncML in Java, Synchronisation von Zeitreporting zwischen Palm Handheld und SAP über den Bea Weblogic J2EE Server.
2000	Praktikum bei den Schweizerischen Mobiliar Versicherungen. Microsoft Access VBA Programmierung und Excel.
1998-1999	Nebenjob in der Cornelia Versand GmbH. Telefonist für Deutsch und Französisch.