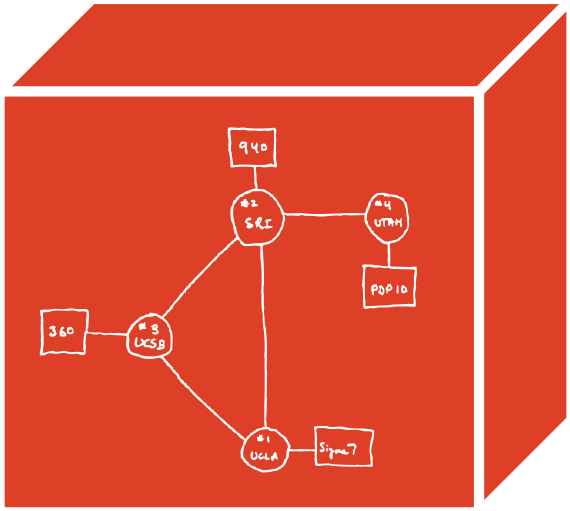
INTERNET FACTORIES



Rudolf Strijkers

INTERNET FACTORIES

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor

aan de Universiteit van Amsterdam

op gezag van de Rector Magnificus

prof. dr. D.C. van den Boom

ten overstaan van een door het college voor promoties ingestelde

commissie, in het openbaar te verdedigen in de Agnietenkapel

op donderdag 13 november 2014, te 10.00 uur

door Rudolf Joseph Strijkers

geboren te Purmerend

Promotiecommissie

Promotor:	prof. dr. R.J. Meijer
Promotor:	prof. dr. ir. C.T. A. M. de Laat
Overige leden:	prof. dr. B. Plattner
	prof. dr. ir. H.E. Bal
	prof. dr. ir. R.E. Kooij
	prof. dr. J.A. Bergstra
	prof. dr. P.M.A. Sloot
	prof. dr. M.T. Bubak
	dr. P. Grosso

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

UNIVERSITEIT VAN AMSTERDAM

Design: Janneke Kors

ISBN 9-789090-285092

For my mother
and my wife Jasmine

Contents

Samenvatting

We leven in een informatiemaatschappij die wordt bijeengehouden door het internet. Het internet begon als wetenschappelijk experiment en is uitgegroeid tot een enorme computerinfrastructuur waarin mensen en objecten informatie uitwisselen, maar de ongekende snelheid waarmee het Internet zich ontwikkelt is een probleem. Technologieën komen en gaan met een snelheid die nauwelijks is bij te houden. Denk aan het verschil tussen de eerste mobiele telefoon en een huidige mobiele telefoons. Mijn huidige mobiele telefoon heeft op Swisscom's 4G netwerk een bijna 7000 keer snellere verbinding met het internet dan mijn eerste 14k4 modem verbinding zo'n twintig jaar geleden. Ook voor de verdere ontwikkeling van de informatiemaatschappij is het internet cruciaal: slimme steden, onderling communicerende auto's en cyber-physical systemen draaien om communicatie met alles en iedereen.

Het gevolg is echter dat computernetwerken in toenemende mate complexer worden en moeilijker te innoveren. Met elke nieuwe ontwikkeling nemen de mogelijke combinaties en interacties met bestaande technologieën en apparatuur toe. Op de lange termijn is dat niet meer vol te houden. Verder is het inefficiënt en te duur om voor elke nieuwe technologie of nieuwe functie een nieuw apparaat te moeten kopen en dat handmatig te moeten inrichten. Er is een innovatiemodel nodig waarmee technologieën kunnen en geïntroduceerd, maar ook kunnen worden uitgezet zonder dat we daarvoor de hardware moeten vervangen.

Deze thesis presenteert de concepten voor een innovatiemodel gebaseerd op software. In plaats van voor elke technologie nieuwe apparaten te introduceren worden netwerktechnologieën geïmplementeerd als applicatiesoftware. Ook de uitrol en beheer van de technologie wordt beschreven door applicatiesoftware. In dit model zijn telecomnetwerken niets anders dan applicaties die draaien op een infrastructuur van servers. Voor informatica leidt het tot een nieuwe researchvraag: hoe kan op een gestructureerde manier netwerktechnologie worden ontwikkeld in applicatiesoftware?

Het resultaat van mijn onderzoek kan worden samengevat in het concept, en daarom ook de titel van mijn thesis, *Internet Factories*. Dit concept beschrijft de stappen om *Netapps* te ontwikkelen en een software platform, de Internet Factory, om Netapps uit te voeren. Een Netapp is een beschrijving van de werking van een netwerk in de vorm van een computerprogramma. Op basis van de Netapp en parameters produceert een Internet Factory

een op zichzelf staand netwerk en de bijhorende structuren voor het beheer. Een voordeel van computerprogramma's is dat een software bibliotheken en ontwikkelpatronen kunnen worden gebruikt om de ontwikkeling van nieuwe technologieën te versnellen met hogere betrouwbaarheid. Experimenten met een Internet Factory proof of concept, Sarastro, laten zien dat een netwerk in essentie niets anders is dan een manifestatie van een Netapp. De contributie van deze thesis kan dan ook worden samengevat als: *de Netapp is het netwerk.*

Het werk in deze thesis is volbracht op een moment waarin de telecom industrie begint netwerktechnologie en diensten los te koppelen van onderliggende infrastructuur en daarom zijn juist nu de resultaten uit mijn thesis relevant. Ik heb in mijn promotie de structurele basis gelegd voor het ontwikkelen van netwerken in de vorm van applicatiesoftware. Opkomende technologieën in de telecom industrie zoals Software Defined Networks (SDN) en Network Function Virtualization (NFV) zijn daarvan de voorbode. Mijn werk laat de bredere context zien waarin software-gebaseerde netwerken kunnen worden toegepast en dat onderzoek in computernetwerken zich zal verplaatsen naar de applicatie domein. Dat is de basis voor een nieuw informaticaprobleem: welke structuren en mechanismen zijn nodig om Netapps te maken die kunnen schalen tot de grootte van het internet? Met andere woorden, wat zijn de limieten van de programmeerbaarheid en beheersbaarheid van Netapps?

Summary

We live in an information-centric society kept together by the Internet. From its humble beginnings as a scientific experiment, the Internet is becoming an immense data processing infrastructure in which people and manmade objects exchange information. However, the incredible pace at which the Internet is developing is a problem. The telecom industry can hardly catch up with the speed of technological developments. Think of the difference between the first mobile phones and todays smart phones. My current smart phone on Swisscom's 4G network offers almost 7000 times more bandwidth to the Internet than my first 14k4 modem connection around twenty years ago. Also for the realization of smart cities, connected cars, and other cyber-physical systems the Internet plays a crucial role, as these developments will interconnect everyone and everything.

The consequence is that computer networks are becoming increasingly complex and difficult to innovate. With each new technology and development the number of combinations and interactions with existing technologies and devices only increases. This is unsustainable in the long run. It is inefficient and expensive to introduce technology as a package of hardware and software, which have to be embedded in existing systems often in an application-specific manner. A new innovation model is required in which hardware, software, and services are decoupled.

This thesis presents the concepts for a sustainable innovation model for computer networks based on software. Instead of introducing devices implementing new network technologies, network technologies are implemented in application software. Consequently, telecommunication networks and services are simply applications running on a networked infrastructure of servers and a new network technology is simply the creation of an application program. This is a new avenue of research for computer science that has not been given attention yet: how to create a methodical process for programming computer networks from application programs?

This thesis contributes a novel concept for solving the presented research problem, which is named *Internet factories*. The Internet factories concept describes the steps to create *Netapps* and the software platform, the Internet factory, to allow its execution. A Netapp is a description of the behavior of a network in all its aspects in the form of a computer program. An Internet factory uses the Netapp and additional input parameters to manufacture and

deploy application-specific Internets including the mechanisms for its operation. A major advantage of this approach is that developers of Netapps can use standardized components, common patterns, and software libraries for the creation of Netapps and profit of higher reliability and faster development times for new network technologies. Experiments with a proof of concept, called Sarastro, show that in essence the network is just a manifestation of the Netapp. Therefore, the contribution of Internet Factories can also be summarized as: *the Netapp is the network.*

The Internet Factories concept comes just in time while the telecommunication industry is becoming aware of the emerging need for mass customization and complete unbundling of network services. In my thesis, I have created a structural basis for the development of computer networks in application software. Upcoming technologies, such as Network Function Virtualization (NFV), Software Defined Networks (SDN), and Openflow, show that the vision of this thesis is becoming reality. My work also shows that the research focus in network research will shift to the development of Netapps, as Netapps are at the basis for articulating a new computer science problem: can we create computer programs that scale to the size of the Internet? In other words, what are the controllability and programmability limits of Netapps?

1

Introduction

Moving to a larger purpose widens
the range of solutions.

Gerald Nadler

1.1 Motivation

Whether the result of careful design, fortunate accidents, or both, the Internet has evolved from an experiment [1] to one of the largest and most complex systems ever built. In some countries, the Internet has now surpassed voice-communications as the major service of telecommunication infrastructures. Billions of people are now using Internet applications, such as e-mail, YouTube, and Facebook on a daily basis. Indeed, the Internet has become an important, if not the most important, means for exchanging information. Still, it marks only the beginning of its essential role in all aspects of the information society, such as smart cities, connected cars, and other cyber-physical systems [1]-[3].

The consequence of these developments is that computer networks are becoming increasingly complex and difficult to innovate. With each new technology and development the number of combinations and interactions with existing technologies and devices only increases. This is unsustainable in the long run. It is inefficient and expensive to introduce technology as a package of hardware and software, which have to be embedded in existing systems often in an application-specific manner. A new innovation model is required in which hardware, software, and services are decoupled.

This thesis presents the concepts for a sustainable innovation model for computer networks based on software. The concepts have been developed to solve three problems in the development of complex distributed systems, such as Youtube or a sensor network spanning cities. The first problem is the diversity of these applications for which network infrastructures facilitate network services. The total network demands for these systems span all dimensions of networks: more speed, more mobility, more connections, and more real-time capabilities [4]. A network infrastructure simply cannot accommodate the peak usage in all dimensions. For example, network infrastructures have to deal with transfer of large files to massive amounts of Twitter messages to real-time streaming of data from e-Science experiments to supercomputers. The configuration of networks to accommodate the diverse network requirements of applications without over dimensioning goes far beyond the capabilities of current protocols and services available to network management systems.

The second problem is that the pace of technological developments is so high that we are approaching a need for sophistication in the development, deployment, and management of network that simply is too complex without information technology [5]. The concept of self-organizing networks (SON) in mobile networks shows that mobile networks are now reaching a level of complexity that only computer programs can determine in which the optimal network configuration parameters [6]. Where it used to be possible to make manual interventions, the speed, scale, and complexity of networks and applications require systems that understand the operating environment and continuously adapt to changes.

The third problem is that current Internet applications must act in harmony with the underlying networks. Internet applications, such as YouTube, are aware of the network environment in which they operate [7], [8]. For example, the performance optimization achieved by the developers of content delivery networks for efficient streaming of videos is to minimize geographical distance between their servers and end-users [9]. Essentially, de-

velopers integrated their application in the Internet for accessing and exploiting details of the network in an application-specific way. In general, complex distributed systems might require control over the network infrastructure normally hidden by the layers of the Internet model.

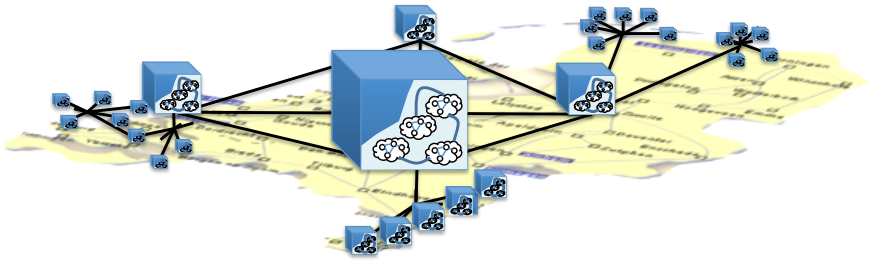


Figure 1. Infrastructure clouds are the basis for future telecom infrastructure. They can be reprogrammed for application-specific exploitation.

This thesis shows that a simple concept can solve these problems (see Chapter 2). In short, an application that programs a network is introduced. The application implements the technologies as well as the configuration and management procedures of a network service. As the application is a program, it can be tuned for specific customers, applications, and network service demands. The assumption, however, is that networks are programmable, such that in the extreme applications can control any aspect of a network. For example, the application might redirect and modify packets or re-configure routers. Looking at the concept from the computer scientist's perspective (see Figure 2), the network is a distributed computing environment in which a developer creates an application program (1) that instructs a collection of network elements (2) to perform specific tasks described in program code (3). Consequently, the introduction of a specific network technology is simply the creation of a new application program. This is a new avenue of research for computer science[1] that has not been given attention yet: *how to create a methodical process for programming networks from application programs?* In this thesis, the required concepts and finally an integrated framework are presented along with their proof of concepts.

The research problem is decomposed into four research questions (Section 1.3). Their solutions are contributions to the thesis (Section 1.4 and Chapter 2-5) and provide the basis of a novel concept for solving the presented research problem, which is named Internet factories (Chapter 6). Finally, the solution to the research questions are presented in Chapter 7.

[1] The University of Amsterdam summarizes computer science as *controlling complexity* [10].

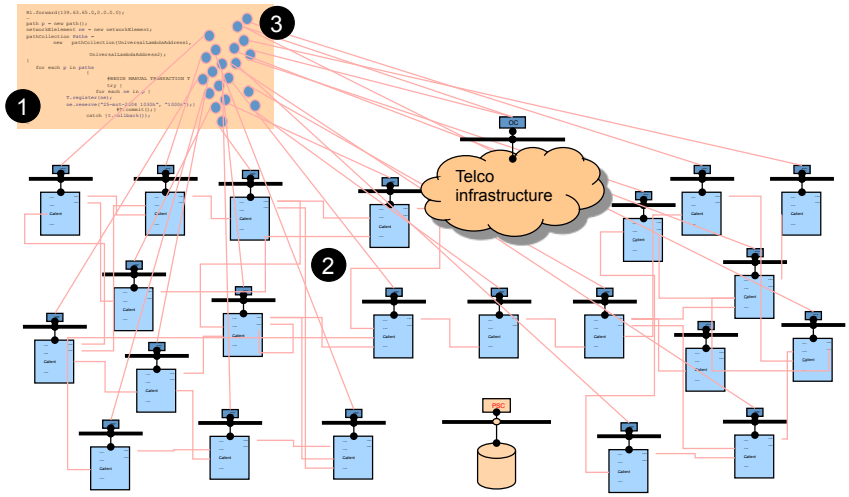


Figure 2. In its elementary form, a programmable network is a collection of network elements that can be programmed to act in a specific way. A developer creates a computer program (1) that instructs a collection of network elements (3) to perform specific tasks (2). The subject of this thesis is to describe a methodical process for programming the collection of programmable network elements.

1.2 Related Work

In its elementary form, a programmable network is a collection of computers, i.e., programmable nodes, which can be programmed in application-specific ways (see Figure 2). Programmable nodes can be loaded with a baseline of functionality, such as the Internet suite [11], but they can also be loaded with application-specific networking stacks [12]-[14]. Programmable nodes can be made to share resources with multiple networking stacks simultaneously [15]-[20]. Programmable nodes can also be used to add new features or to adapt to new operating conditions. These capabilities are possible because network behavior is implemented in software.

Programmable networks enable management of network resources, which specialized single-purpose network devices do not offer. With programmable networks, Internet service providers can tailor software to deliver application-specific network control and automation [22]-[25]. Network can also include application servers and content for applications that are easily affected by small disturbances in the network. Augmented reality [21] is such a use case in distance between servers and end-users impact the quality of experience. With programmable network elements, networks can be efficiently deployed, scaled, *and* removed, making optimal use of the infrastructure resources [26]-[28]. In traditional telecom, migration and decommissioning of legacy networks and services is often a problem. Finally, programmable networks are a solution when flexible deployment and

repurposing are required [30], [31]. In sensor networks, energy constraints and highly specific application domains prevent sophisticated layered architectures found in telecom and the Internet, witnessed by the plethora of protocols and architectures developed over the years [29]. Though programmable network research had considerable interest over the years, researchers explored the design space of programmable networks rather than their use.

The primary goal of major programmable networking frameworks was to expose a set of programmable features to network designers rather than application developers. *Active networks*, for example, enable customization of programmable nodes by loading and executing specific components and functions referenced by third parties via tags or code injected in packets [32]. The approach of *clean-slate networks* prefers programmable nodes in which behavior is determined by an interconnection of objects [12], [33], [34]. Rather than using sophisticated programmable nodes, *Software Defined Networks* (SDN) introduced a centralized programmable component that instructs network nodes [35]. In these developments, the exploitation by applications for application-specific services received little attention.

To exploit the flexibility of programmable networks, either network protocols and services need more knowledge about the applications or applications need more knowledge about network protocols and services. This inevitably leads to more complexity, as more interfaces are required for information exchange and control. For example, employing sophisticated security policies [36], packet filtering [37], and quality of service capabilities [24], [38] also requires more knowledge about applications (e.g., recognizing voice and web traffic) in addition to control over topology, paths, and link qualities. Paradoxically, such crosscutting concerns can only be dealt with when the network engineer understands the application. Only then network engineers can program the network with expert systems and artificial intelligence [6], [39]. From the developer's perspective, this paradox applies as well, as choices about where to place application code, what traffic to filter, where to redirect traffic, and how to recover from failures all depend on network state.

1.3 Problem Statement

This thesis presents the concepts in which distributed applications *program* network services (see Figure 2). To solve the paradox of crosscutting concerns, the problem is to find a concept that can be understood by both network engineers and application developers. Additionally, it must be assumed that in general not all nodes in a network can be programmed. This is due to multi-domain constraints, legacy networks, transparent overlays, multi-layer networks, or simply because there are no interfaces to exercise the required control over intermediate networks, such as with legacy networks. Hence, the entire programmable network design space is bounded by such limitations. Developers must acknowledge such limitations as part of their programming problem. In short, the problem is:

1) *What are the common patterns for the design of applications that use programmable networks and how can these patterns be described in an architectural framework?*

Conflicts and feature interactions might occur when multiple programmers have control over a programmable network. An obvious solution is to create separate programmable network infrastructures for each application domain. In terms of cost, open-endedness, and sustainability, it is better to create a network infrastructure in which resources can be shared than to create a dedicated single-purpose network infrastructure. Sharing the programmable network infrastructure requires some elementary functions for managing resources amongst multiple users typically provided by resource manager. So, the management of programmable network resources leads to the second research problem:

2) *How can programmable networks support multiple application programs that control networks in application-specific ways?*

When developers program a network in a specific way, they must also implement its management. Essentially, the operational boundaries of the (programmable) network determine the tolerance of the application to disturbances in the network. In general, network management systems ensure that the network operates within operational boundaries, which are determined by the network operator. However, the range of interventions to compensate events, such as congestion, errors, and failures, differ from application to application. If developers implement network interventions as part of an application-specific controller, they have to be aware of the impact of interventions on the network and application. This leads to our third research problem:

3) *How can programmable networks be controlled from application programs, and how do controllers affect application design?*

Clearly, programming a collection of network elements from scratch without support from frameworks and libraries, such as for the Internet suite, is a daunting task. State of the art network programming libraries has made distributed programming so easy that developers can forget the technologies facilitating network communication. Any developer that takes on the task of programming a network beyond the Socket API is confronted with an enormous increase in complexity and crosscutting concerns. Even if developers decide to create a new approach to networking, a clean-slate solution for example, they would still have to implement (all or part of) the network functions described in the OSI reference model [40]. These network functions are elementary for remote communication. For practical use of programmable networks in application programs, the task of developing, deploying, and managing programmable networking solutions must be simplified. In other words:

4) *How to structure the development, deployment, and management programmable network applications that make crosscutting concerns manageable?*

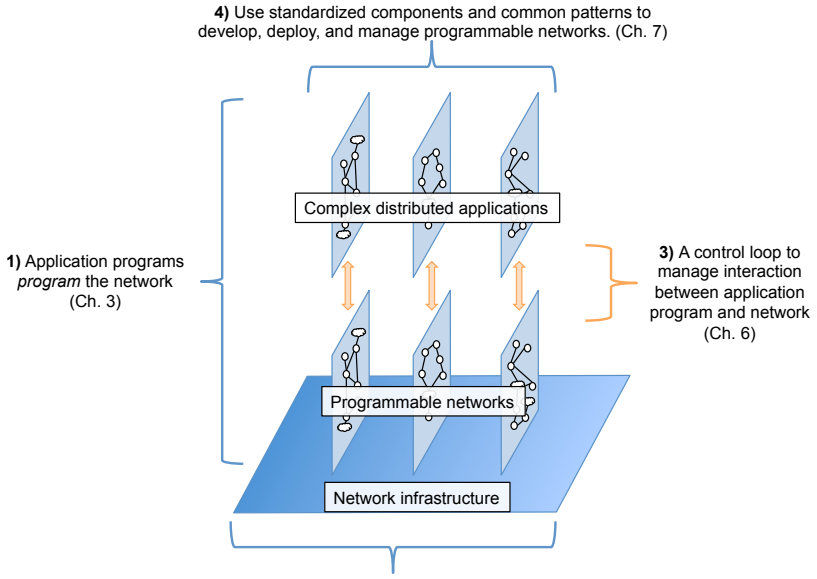


Figure 3. The concepts presented in this thesis required for solving the research problem.

1.4 Contributions

This thesis introduces four concepts that are required for solving the presented programming problem (Figure 3). The first contribution is an architectural framework, called *User Programmable Virtualized Networks (UPVN)* (Figure 3, 1), which defines elementary components spanning the design space of the programmable networks and shows how to create programmable networks from application programs. For instance, the initial UPVN prototype showed for the first time how application developers could control a collection of programmable nodes from a program. The second contribution is *generalized Token Based Networking (gTBN)* (Figure 3, 2), which provides a solution to partition a network infrastructure into multiple distinct programmable networks. These distinct programmable networks are associated with specific network services and user groups. The third contribution is an *Application Framework for Network Control* (Figure 3, 3), which introduces an elementary control loop application for application-specific network management. The final contribution is the *Internet Factories* (Figure 3, 4) concept that provides a methodical process for programming computer networks from application programs. In the following sections the contributions are described in more detail.

1.4.1 User Programmable Virtualized Networks

Chapter 2 introduces the concept of User Programmable Virtualized Networks (UPVNs). UPVNs deliver application specific services using network element components that developers can program as part of an application. In Chapter 2, UPVN's architectural framework is described and it is shown that its development is application driven, creating only those facilities in the network crucial for applications. The proof of concept presented in Chapter 2 and its implementation in *Mathematica* [41] also shows for the first time how an enormous wealth of mathematical, computational and visualization software can be applied to solve application specific network issues.

Figure 4 shows the first UPVN prototype in which programmable network elements are controlled from an application, in this case Mathematica which is a scientific computing environment [42]. The visualization shows the network layer topology of a 13-node programmable network developed earlier [43]. The visualization was the result of a breadth-first search algorithm implemented in Mathematica, which was executed on the programmable network.

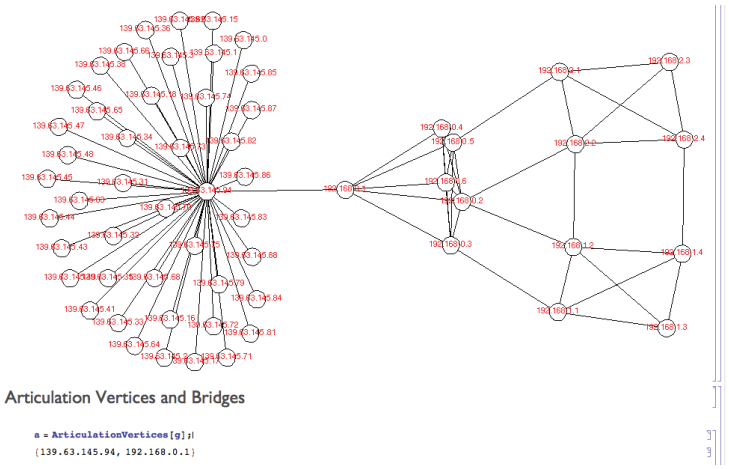


Figure 4. The first UPVN prototype enabled a Mathematica, a scientific computing environment, to control a 13-node programmable network. The screenshot also shows that topology matters can be dealt with algorithmically (`ArticulationVertices[g]`, see Chapter 2,5 and 6).

1.4.2 Generalized Token Based Networking

The UPVN concept shows that networks services can be implemented as applications that program a network. When the network is shared, user traffic might belong to different applications, so there is need for a protocol independent way to bind applications to the specific network services delivered by these applications. The Generalized Token Based Networking (GTBN) architecture presented in Chapter 1 enables dynamic binding of user groups, which can be a group of users or a collection of applications, to application-specific

network services in their own programmable network environment. GTBN uses protocol independent tokens to decouple authorization from time of usage. The use of tokens also allows a third party to act as a resource manager for provisioning application-specific services. The initial implementation of GTBN extended the network stack on hosts and demonstrated for a number of distributed applications, such as MPI and GridFTP, how applications can dynamically bind to a network service.

The Interactive Networks prototype (see Section 1.4.5) implements the GTBN architecture using packet-processing graphs. The packet processing graphs can be manipulated by a multi-touch user interface shown in Figure 5. Figure 5 also shows how packets are processed. Received packets (`netin`) are filtered by token (`tbs`) before further processing takes place. The (`tknlss`) component allows normal IP traffic.

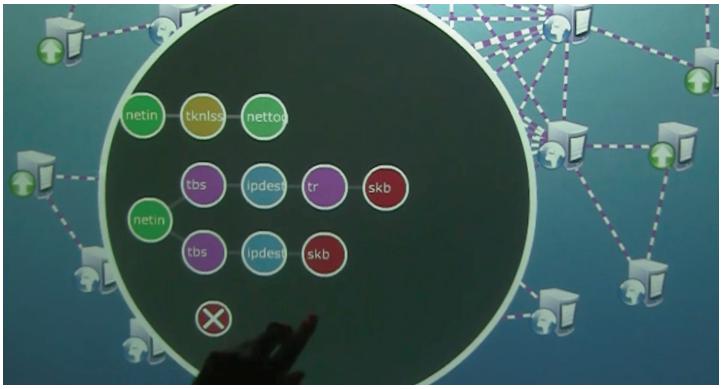


Figure 5. In Interactive Networks, packet-processing graphs isolate application-specific behavior. The packet-processing graphs can be manipulated online via the multi-touch user interface. Interactive networks uses token networking to allow traffic manipulation on a per packet basis.

1.4.3 Application Framework for Programmable Network Control

Errors, failures, and unexpected events can be expected in any information-processing environment. Therefore, UPVNs require the ability to implement measures against events that impact network and application services. Different from the operation of autonomous routing protocols and traditional network management systems, the automated interventions of UPVNs are the result of application logic. An application framework for programmable network control was developed that integrates the control over programmable network elements with application logic. Consequently, the application framework shows how network events impact application logic and how specific knowledge of network protocols and services limits scalability of application logic. This leads to a new and fundamental scientific problem on how to create programs with predictable robustness and adaptability, which is considered future work (see 7.2).

Figure 6. Amazon EC2 and Brightbox Infrastructure-as-a-service clouds were used to deploy UPVNs. In one instance, a 163-node UPVN was created using all the 18 data center locations of the global cloud infrastructure.

1.4.4 Internet Factories: Creating Application-Specific Networks On-Demand

The Internet factory concept is based on the broader concept of software factories in software engineering. The rationale of software factories is that the majority of application development can be captured by standardized components and common patterns in software libraries. Similarly, Internet factories enable developers to re-use and customize known solutions in the design and implementation of programmable networks and to make their own solutions available to other developers. Moreover, years of knowledge and best practices in systems and network engineering can be made available via software libraries.

Internet factories show how to create UPVNs on top of the Internet even if network operators do not facilitate the programming of their routers. For example, a 163-node UPVN was deployed on the 18 locations of Amazon EC2 [44] and Brightbox's [45] global cloud infrastructure. This proof of concept shows that the concepts and technologies that were developed in this thesis can already be applied today. With the use of cloud-based network elements (see Figure 6) and the Internet, UPVNs can be created and deployed while taking into account administrative domains, legacy networks, and limited network programming interfaces. Summarizing, cloud-based networks mitigate the lack of full control over the network infrastructure in a way that was not possible with programmable networks (such as active networks).

The interfaces required by Internet factories can be implemented in existing networks or provided by infrastructure-as-a-service clouds [27], [46]. Therefore, its concepts can be applied to solve problems in current networks. For example, this thesis led to three patent applications for creating application-specific network optimizations in mobile networks [47], fixed networks [48], [49], and end-user devices [50].

1.4.5 UPVN Prototypes

A number of UPVN prototypes have been created of which three are described in this thesis (Chapter 1,4 and 6). The initial UPVN proof of concept was developed on a 13-node computer network consisting of desktop computers and a customized Linux kernel. The first UPVN prototype was demonstrated at the Dutch Exhibition booth, Super Computing 2008, Austin, Texas, and its design and implementation later published [51], [52]. The proof of concept, called Interactive Networks, consisted of a network implemented by VMware virtual machines running a UPVN. The proof of concept allowed users to create and manipulate network paths interactively using a graphical user interface on a multi-touch table (see Figure 7). Interactive networks illustrated how networks existing only in software can be controlled from the application domain. Subsequent experiments brought network control to other application environments, such as scientific computing [41]. Interactive networks was also integrated into a Grid [53] workflow management systems to allow management of network resources from workflow-enabled applications (presented in Chapter 4). The proof of concept using scientific workflows was demonstrated at Super Computing 2009 in Portland, Oregon.

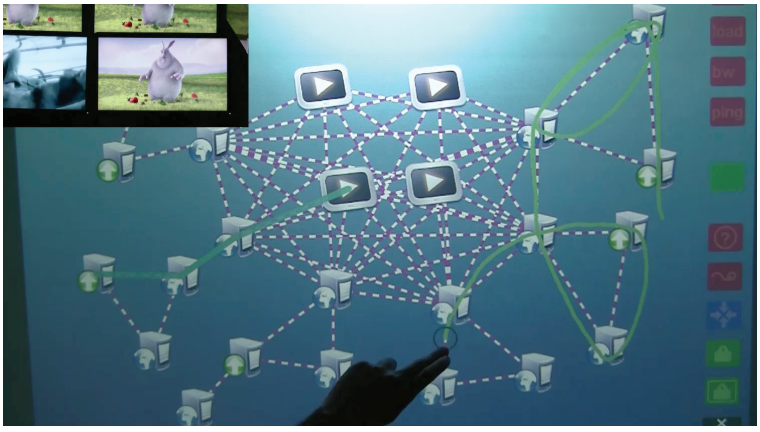


Figure 7. Interactive networks enables users to drag paths between video stream producers and screens displaying the streams. UPVN supports application-specific behavior, such as drawing loops, creating a specific chain of nodes to traverse for packet flows.

The early experiences and collaborations with developers of scientific workflows showed that programmable networking concepts fit well in Grid architecture, as scientific applications already interact with workflows and Grid to request and configure resources. Eventually, the whole Grid infrastructure was encapsulated in virtual machines. A prototype was developed capable of bootstrapping tailor-made Grid infrastructure for individual applications [54] and complete workflows [55]. Since the Interactive Networks prototype was encapsulated in virtual machines as well, this experience could also be applied to bootstrap-

ping of networks. Figure 8 shows a discovery step of the prototype developed in 2010. A part of the process of booting Interactive Networks is the discovery of the virtual network infrastructure. The creation and deployment of an interactive network with arbitrary network size and topology was implemented in an application program. The prototype was demonstrated at the Dutch exhibition booth at Super Computing 2011, Seattle and the 3rd International NGN workshop 2011 in Delft, the Netherlands. Practical experience in the design and implementation of the presented concepts, frameworks, and architectures shaped the development of Internet factories.

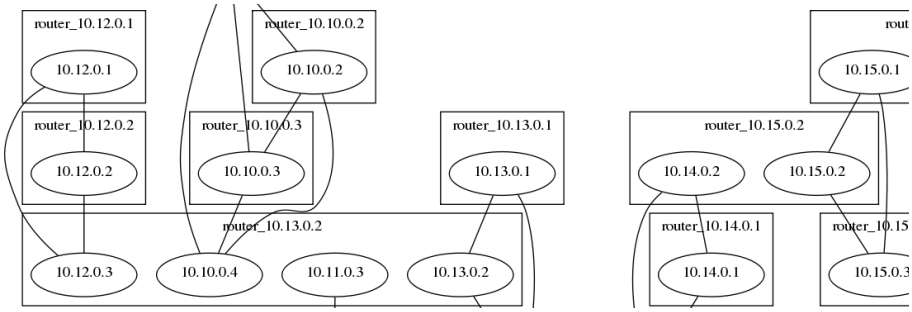


Figure 8. A part of a visualization of Interactive networks while it is discovering the virtual network infrastructure.

1.5 Overview of Work

Research started from the idea *the network is a program*. Long before the academics and industry invented Network Function Virtualization (NFV) and Software Defined Networks (SDN), the work in this thesis already published the practical feasibility of their underlying concepts [43]. One of the challenges of *the network is a program* idea was to move from programming single network nodes, such as with Netgraph [56] and Click [57], to the programming of a collection of network nodes from a single program. My exploration of the programmable network design space started with programming commodity servers with Ruby programs (capable of controlling Linux packet processing). Later, also experiments with programmable sensor network nodes were done. The choice for sensor network nodes was driven by research on sensor networks, which led to new programming paradigms for programming a collection of sensor nodes, e.g., Macroprogramming [30].

One of the results of sensor network research, which we also used for design space exploration of programmable sensor networks, was Java Sun Spots [58]. Java Sun Spots were programmable network nodes with a wireless connection and numerous sensors, such as a thermometer. A complete Java integrated development environment supported application development and enabled rapid prototyping and design space exploration, which was done from 2009 to 2011 by a number of students and as part of a master course *Advanced Networking* [59].

The originally developed scriptable Ruby environment for packet processing was further extended with Streamline [60] supporting packet manipulation at line speed on physical machines. However, programming a network of physical computers leads to practical problems. For example, the development cycles were long because a single bug would crash the whole network and require manual reset. Also Openflow [22] and ForCES [61] were considered for programming networks, though these technologies were too limited for developing a reference implementation of the UPVN architectural framework. Further work explored the use of virtual machines. Virtual machines allow manipulation of networks beyond the state-of-the-art of physical network elements.

The most straightforward manner to explore the use of virtual machines was to create a controller for deployment and configuration of existing prototypes on virtual infrastructures, such as Infrastructure-as-a-Service clouds. The development of such a controller for UPVN led to both conceptual and practical problems, amongst others: network management, location selection, dynamic user-network interfaces, multi-domain interoperability, and virtual machine image distribution. The design space for cloud-based deployment and configuration of programmable networks was explored in student projects and TNO research projects. In 2012, I developed a cloud-based prototype of a programmable network infrastructure to prove that networks could be created, deployed, and managed from computer programs. Essentially, the prototype combined all the previous work and resulted in the Internet factories concept. KPN, the Royal Dutch Telecom Operator, Long Term Research program also allowed further refinement of cloud-based network design and deployment. The architecture and implementation of the Internet factories concept establishes that the essence of a network is a Netapp. A Netapp is a program that determines the behavior and life cycle of network nodes, e.g., their interactions, and interventions to compensate for errors and failures. In Internet factories, the network is just a manifestation of the Netapp and research continues on its design and implementation. So, it's not about programmable network nodes anymore, but about the programs that created them.

A timeline of the work is shown on pages 24-25.

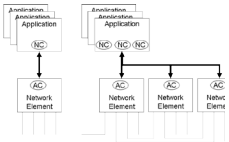
1.6 Thesis Outline

User Programmable Virtualized Networks (UPVNs) and its initial prototype are described in Chapter 2. Then, in Chapter 3, the generalized Token Based Networking (GTBN) architecture is presented. Chapter 4 describes how these concepts can be applied to workflow management systems to control networks for specific applications and user groups. Experience and lessons learned from design and implementation of the prototype lead to an application framework for network control, which is described in Chapter 5. Chapter 6 describes the Internet Factories concept that simplifies the creation and deployment of user programmable virtualized networks. Finally, conclusions, future work, and an outlook are presented in Chapter 7.

From applications program the network ...

2007

User Programmable Virtualized Networks (Ch. 3)



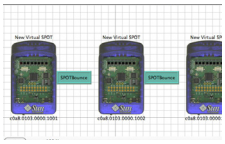
2008

Integrating Networks with Mathematica (A-14)

Interactive Networks (P-6)

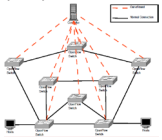
2009

SpotSim: Test Driven development of Sensor Network Applications (T-10)



Programmable Interplanetary Networks (T-12)

Self–Adaptive Routing (T-7)



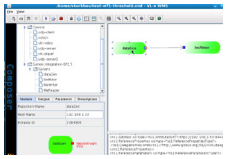
Interactive Control over a Programmable Computer Network Using a Multi-touch Surface (A-12)

Supporting communities in programmable grid networks: gTBN (Ch. 4)

2010

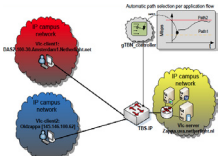
Application-Specific Routing in Sensor Networks (T-11)

Network Resource Control for Grid Workflow Management Systems (Ch. 5)



Intercloud Operating System (P-4)

Generalized Token Based Networking (P-5)



The citations between brackets are references to overview of work shown at the end of the thesis.

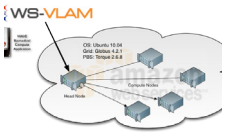
- MSc/BSc thesis
- Publications
- Research
- Patents
- Super Computing Demonstrations

to the *Netapp* is the network.

2011

Network Performance in Virtual Machine Infrastructures (T-9)

Grid on Demand (T-8)



AMOS: Using the Cloud for On-Demand Execution of e-Science Applications (A-11)

2012

Bootstrapping the Internet of the Future (T-6)

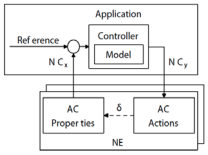
Operating Systems for On- demand Networks (T-3)

Towards Executable Scientific Publications (A-9)

A Cloud Storage Platform in the Defense Context - Mobile Data Management with Unreliable Network Conditions (A-5)

Sun Tzu's Art of War for telecom (A-6)

Application Framework for Programmable Network Control (Ch. 6)



Cloud-Based Interactive Networks On-Demand (P-3)

2013

Improving Quality of Experience of the Internet Applications by Dynamic Placement of Application Components (T-2)

Robust Applications in the Open Internet (T-5)

Discovery and Query Service for Distributed Clouds (T-4)

Architecture of Dynamic VPNs in Openflow (T-1)

Defining Inter-Cloud Architecture for Interoperability and Integration (A-7)

Towards an Operating System for Interclouds (A-2)

Localization and placement of servers in mobile networks (O-2)

PDN-GW redirection for dynamic service delivery(O-3)

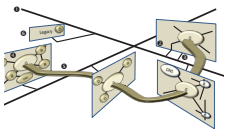
Client-driven resource selection and allocation for server workload (O-1)

2014

Defining Intercloud Federation Framework for Multi-provider Cloud Services Integration (A-4)

MeTRO: Low Latency Network Paths with Routers-on-Demand (A-3)

Internet Factories: Creating Application-Specific Networks On-Demand (Ch. 7)



2

User Programmable Virtualized Networks

Program services instead of re-architecting the network and the management system for every new service.

Peter Lothberg, Internet Legend

Meijer, R.J.; Strijkers, R.J.; Gommans, L.; de Laat, C., "User Programmable Virtualized Networks," *e-Science and Grid Computing, 2006. e-Science '06. Second IEEE International Conference on* , vol., no., pp.43,43, Dec. 2006.

As second author, I have contributed to the concept and architecture, and implementation.

This chapter introduces the concept of a User Programmable Virtualized Network, which allows networks to deliver application specific services using network element components that developers can program as part of a users application. The use of special tokens in data or control packets is the basis of a practical, yet powerful security and AAA framework. The concept allows for implementations with a low footprint that can operate in a multi domain network operator environment. We demonstrate the ease with which one can build applications and address networking problems as they appear for example in sensor networks.

2.1 Introduction

The concepts defined in the OSI model for the interaction between networks, end systems and their applications, are widely accepted [11]. International telecommunication infrastructures and the Internet are based on these concepts. Because details like network topology are irrelevant to most applications, OSI considers only end-to-end transport services.

What if network providers cannot understand all of your network service demands anymore? What if the network cannot be over-provisioned due to the involved costs? If one detects that an IP router will fail shortly, how can we route a VoIP stream over an alternate path before the router actually fails and before the users notice anything? Do video streams of a burning car have priority over those of collided cars not far away in a heavily congested network? In such cases, there is a need to tune the network service to the demands of the users and their application programs; one has to facilitate application specific networking.

Neither the set of Internet protocols, nor a network management system (NMS) provides practical control interfaces to individual network nodes. The services of TCP and UDP are often used trough socket APIs. Socket APIs however, hide most of the network details such as topology. In theory, using the NMS would be one way for the application programmer to discover and possibly control network elements, such as Cisco's Active Network Abstraction [62]. The span of control of an NMS however, is typically restricted to a single network operator domain. Furthermore, NMSs are designed to support operators and not end user programs. Moreover, only operators, not end users, are allowed to use the NMS.

The concept of programmable networks is sufficiently well known to create concepts and technologies that support application specific networking [14]. The concepts differ in how applications interface with network nodes. Basically there are three variants: agents, active messages (also known as active networks) and remote method invocations (RMI) [63]. In short, agents are programs that travel from node to node, active messages are network packets extended with application code and web services are a great example of RMI. The IETF ForCES working group standardizes common elements in IP routers [64]. One of the benefits is that elements may be changed or added and combined with web and Grid services [65], [66]. ForCES does not have a security concept that is practical in a multi domain network (see Section 2.3). Currently, years of developments in programmable networks have lead to complex frameworks with corresponding complex technologies. This prevented the emergence of killer applications and market impact [67].

Sensor networks are frequently designed to operate in a very dynamic context, in which sudden environmental changes may cause parts of the network to become isolated. This has inspired the ad-hoc networking concept, where a system of identically programmed sensors collectively supports a telecommunication service amongst themselves [68], [69]. Research, however, is predominantly focused on topics as autonomy and self-organization of sensor systems [69]. Little, if any, attention is given to the interaction between end-user applications and the sensor network and to the fact that in realistic situations sensor networks belong to multiple organizations.

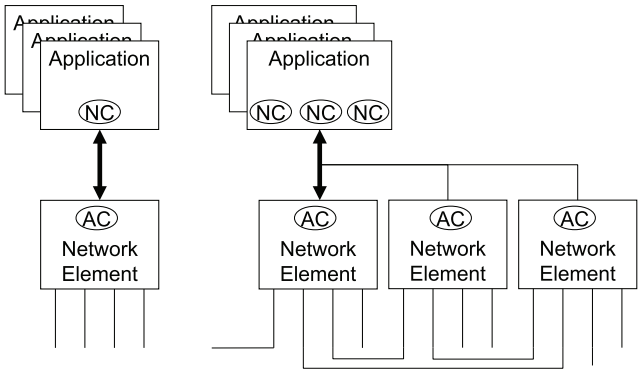


Figure 9. UPVNs model for the interworking of a network element and applications. Left: the virtualization of a network element (NE) as NE Component (NC). Right the virtualization of a network. The NEs are either interconnected directly to each or connect to the Internet (open lines).

2.2 User Programmable Virtualized Networks

User Programmable Virtualized Networks (UPVNs) is a concept that allows end-user applications to interact closely with network elements (NEs) such as IP routers. Ultimately UPVN concepts should be applied in sensor networks where technologies with small footprints are required. Therefore, UPVN's development is application driven; creating only those facilities that are crucial for applications. Yet, UPVN uses much of the concepts presented in [65], [66]. Most importantly, using Grid concepts, one can regard individual NEs as resources, which are exploited through the Internet as components in application programs. A NE component (NC) can be seen as a manifestation of the NE in the application, i.e. a virtualized NE. Consequently, all virtualized NEs together create a virtualized network, allowing interaction with user programs, as shown in Figure 9. Hence, our concept is named User Programmable Virtualized Network or, abbreviated, UPVN.

For telephone conversations, the network access provider is responsible for setting up the end-to-end connection. An Internet service provider is used to connect to the Internet. For application specific networking, this is generally not possible. NEs along a path can belong to different network owners. Technical and other (e.g. financial) conditions to access and

use NEs along the path may differ such that the access provider cannot judge if NEs are good enough for the application. The application must make its own judgments. We describe in [70], that in multi domain situations, the use of tokens provides a practical and robust security and AAA framework.

A token can be considered as a reference to a service the user has agreed with the network operator. An application can interact with the NC in order to reach this agreement. By subsequently allowing applications to insert these tokens in a secure manner into the data or control stream of a network, applications are able to signal the NEs that it is authorized to use the agreed service and to be accounted for it. Compared to the alternative, the integration of back office systems of network operators, a token based security and AAA mechanism is easy to implement, see Section 2.3.

2.2.1 Virtualized network element

Figure 9 shows UPVN's interworking model of a NE and a network of NE's with applications. The NE uses technologies, such as Grid- and web services, to expose interfaces on the Internet. Through the interfaces a NE exposes, various applications interact simultaneously with the NE. As such, each application is capable to optimize the behavior of the NE accordingly. During the design phase of the application, the NE appears as an object, called a network component (NC), in the development environment. During run time, our model, as well as state of the art technology, allows dynamic extension of the set of NEs the applications interacts with.

To accommodate application specific packet processing, to set particular parameters of the NE, and to facilitate other functions NEs play in a UPVN, NEs have the ability to deploy application components (ACs). ForCES [64] and the Click Router [57] deploy similar concepts. In UPVNs, the delivery method of ACs to the NEs is regarded to be application specific. As one of the available methods, one could use the NC to deliver the AC to the NE. In any case, care should be taken to avoid problems caused by ACs acting on the AC to NC network traffic, e.g. by using tokens.

ACs can either operate directly on the packets, or sent them to the NC. Then, through the NC, the application can manipulate the packet and sent it back. As an interesting consequence applications can also be constructed that transfer the received packets via any of the NCs into another NE. This NE can put the packet via an AC on a network link or into the IP routing engine of the NE. We have called this process "packet warping", as packets disappear in one NE from the network, to reappear in a supposedly disjoint NE somewhere else.

2.2.2 NEs and the Internet

NEs have IP routing capability and can be connected to other NEs and the Internet. Consider Figure 10, which shows eleven NEs. In UPVN NEs can disappear and reappear, similar to what happens in mobile sensor networks. The Internet is depicted as a network cloud,

appearing several times in Figure 10. NEs like 3, 7 and 8 have direct connections to other NEs (1, 4 and 5) via links that are completely under NE control. They may carry Internet traffic, but a NE could exert manipulation and control on it.

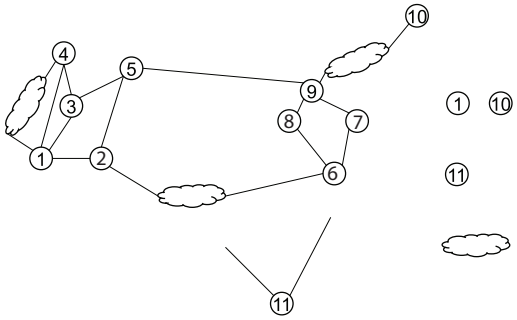


Figure 10. UPVN regards two kinds of links: those completely managed by NEs (straight lines) and Internet links (clouds).

Links such as between 9 and 10, that are not under control of a NE pair, carry in our model regular, best effort Internet traffic. NEs like 1 and 4 are linked also via the Internet. NE 10 is reachable only through the Internet. NE 11 is not connected at all yet, but could be, either directly or through the Internet. A manual or application driven action could be responsible for this.

Another way to look at Figure 10 is that the Internet is encapsulated with NEs. Because of this the Internet is implicitly virtualized in UPVN, allowing applications a specific multi-point interaction with the Internet. UPVNs approach to application specific networking has deep impact on the behavior of NEs. The processing of packets is now, opposite to IP routers, inherently stateful: two identical packets sequentially send to a NE may be processed differently: for example processing may stop if the stored value is depleted in a pre-paid account.

Important questions are: "What functions of the NE should be exposed in the NC?" and "What functions should ACs have?" On basis of our experiences with virtualizations of optical switches [71], we would choose for the ability to intercept, re-route, and reserve resources such as transmission lines. It is however hard to make choices. For some applications, control over the TCP maximum segment size is crucial. Others would like to use a certain transmission path if the price is right at the right moment. In this paper we focus on important service aspects such as forwarding, transactions, topology and quality.

2.2.3 Token based NE services

Both the OSI model and the IP define various Transport and Link Layer services. Together they create end-to-end services. Hence packets do not contain fields explicitly designed to

trigger the invocation of specific AAA mechanisms and the correct NE services. In [70] we describe a secure token mechanism, which we use in UPVN in cases where NE services or the packets themselves have to be secured. In UPVN, tokens can be added for example to the IP options field.

Tokens can be cryptographical products using a key issued by the application. The result is placed in the IP options field, whilst the key is securely provisioned via a NC to a "Token AC" in the NE. Cryptographic operations of the Token AC validate the token. Logic in the Token AC selects the proper AAA AC and has subsequently other AC services executed. If a packet does not contain tokens, if tokens are not recognized, or if no ACs are configured who operate on regular IP packets, the packet follows the default IP processing in a NE. Operators of different network domains only need to understand the cryptography of the token and its associated key and do not have to integrate each others AAA and trust systems [70]. For this reason, UPVNs use of tokens can ignore the discouragement of the ForCES working group to create access to NEs for end-user applications from other network operator domains. Finally, security and AAA issues related to the use of the NC mechanism could be addressed by Grid and web service technologies, although in the case of sensor network applications, they may require too many resources.

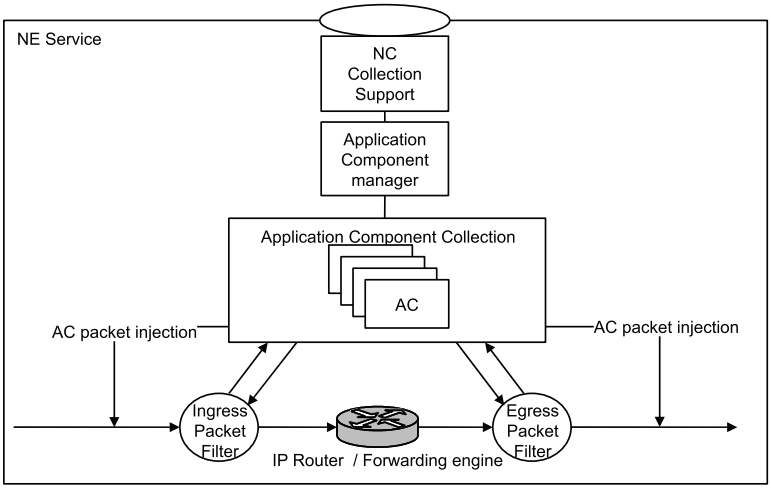


Figure 11. Overview of the NE implementation in Linux.

2.2.4 Implementation of the NE service

To gain experience, and to develop the UPVN concept toward small footprint implementations, a prototype NE was implemented. This NE is basically a PC with Ethernet cards running the Linux OS. The OS was modified to add functions in kernel and user space. Figure 11 shows the architecture of the NE. The design is straightforward: we have encapsulated

the in- and outputs of the Linux Kernel IP routing and forwarding functions with facilities to filter and to inject packets. The packet injection and filter facilities use the netfilter function `libnetfilter_queue` (the successor of libipq). `Libnetfilter_queue` acts both in the kernel and user space of the OS. ACs, which execute in user space, can use it to exercise detailed control over the IP traffic. Packets can be dropped, generated and modified by the AC and, through the AC-NC connection, even by applications. ACs are stored in the Application Component Collection (ACC). By manipulating the ACC, the Application Component Manager (ACM) controls the lifetime of an AC and the tree like chaining of AC. This enables ACs to operate on one or more copies of packets or packets already modified by ACs.

The NC Collection Support module (NCS) takes care of the NE service interface and contains functions to identify the NE and other functions that are helpful in case the NC becomes part of collections in applications.

Amongst others, NCS exposes the AdjNe AC that discovers the neighbors of a NE and identifies them as an NE or regular Internet node (router, end system).

The user-space programs in the NE are coded in Ruby [72], a language allowing an efficient first-time implementation of UPVN concepts. With Ruby's DRb (a remote method invocation facility) the NE services interface was created, avoiding for the moment the complexities and consequences of web services on a NE: parsers, web servers and performance implications [67]. Furthermore, it allowed exploring network service creation with clear concise Ruby programming. In Section 2.4.1 the use of web services is discussed.

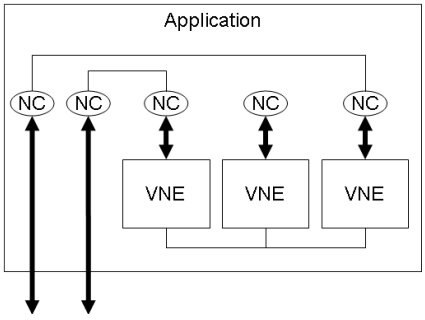


Figure 12. Because of the NC that hides details of the NE implementation, applications are unable to discriminate between real and virtual NEs (VNE) that are application modules.

2.3 Virtual NEs and Virtual Links

As stated in Section 2.2.1 in UPVN it is possible to warp packets via the application between NEs. This, and the fact that the NC shields details of the NE, allows the existence of virtual links and virtual NEs, which reside inside the application.

Figure 12 shows three virtual network elements (VNEs). These application components may function such that it appears to have network connections between both to each other and to real NEs via NCs.

VNEs can be implemented in any way the application developer seems apt. It is rather straightforward to imagine that VNEs run the same algorithms as NEs. Indeed, VNEs might contain complete IP routing functionality even with assigned IP addresses and as such bring the Internet into the applications.

The concept of VNEs inspires to even more exotic constructs. The application developer is free to include NCs of NEs and VNEs in a VNE, the start of recursive pattern. A potential very useful application is one in which the application developer pairs a VNE with a real NE. Parameters indicative for the performance of the NE are obtained by the VNE through an incorporated NC. Applications may then interact for certain matters with the VNE. e.g. if the NE reports its links to the VNE, topology inquiries can be done to the VNEs.

2.4 The Services of Virtualized Networks

An application, which contains one or more NCs, is said to contain a virtualized network. An application program might contain only the NC of single NE even if the network contains hundreds of other NEs from which NCs can be obtained. Regardless if an application contains only one or all NEs, we say the network has a manifestation in the application. There are good reasons for manifestations with a single NC, for instance the filtering of traffic at a strategic point. Therefore, details of manifestations of networks are in UPVN considered application specific, requiring no general frameworks, facilities and structures. UPVN lacks general discovery services, brokers, billing services, AAA servers, etc. The usefulness of this depends on the application. In sensor networks, for instance, there might just be not enough infrastructure around to support an elaborate framework.

The absence of a general framework is not only contrasting to major trends in programmable networks [14] but also to prior developments by the TINA consortium of leading telecommunication vendors, Telco's, and their research institutes. They modeled the network and administrative facilities of a traditional telecom operator [73] in an object oriented, CORBA based [63], framework.

The details of the interfaces between applications and NEs are also application specific. Depending on their insights, developers choose different implementation technologies for a given application. As stated in Section 2.2.4, the Ruby RMI DRb was currently considered to be more suitable to implement a NE service interface then web services. The footprint of web services in the NE was expected to be too large.

In this thesis, we investigate the consequences of the network virtualization itself. We do not want to standardize interfaces or invent brokers, directory services, etc., before experience, insight and scientific studies yield reasons to do so.

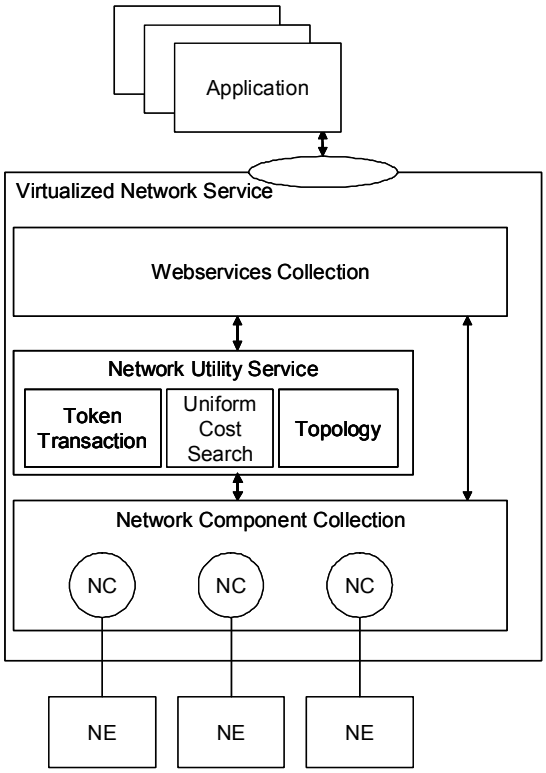


Figure 13. Architecture of the virtualized network service. This architecture is specific for topology aware applications as the ones we build in Mathematica.

2.4.1 The implementation of UPVN

We have created several applications of network services following the UPVN concept; some of them are described in the next sections. The service architecture of Figure 13 followed from our applications and experiments. We have created a Virtualized Network Service (VNS) that exposes a SOAP based web services interface. Here, its large footprint is not considered to be a problem, since the VNS runs on a dedicated computer and not on a NE with limited capabilities as in the case of sensor networks.

The combined services of the Network Utility Service (NUS) and the Network Component Collection (NCC) can be used via the VNS. The NCC provides access to specific ACs. Figure 13 illustrates that NCs have (Ruby specific) connections to the NE and more specifically to the NCSs service interface (see Section 2.2.4). Applications and NEs can load the NEs with specific ACs during run time using the ACM (Section 2.2.4).

The NUS was developed for applications that use path and topology functions to manipulate certain token-tagged streams. The Token Transaction Service (TTS) allows for transactions on reservation of NE packet forwarding capacity between specific incoming (ingress)

and outgoing (egress) links of specific tagged IP packets. The TTS controls a specific AC for this. The Uniform Cost Search Service (UCSS) finds the least cost path by using the `AdjNe` AC described in Section 2.2.4. AdjNe is also used by the Topology Service (TS) to discover NEs and Internet access points in a UPVN. The TS is also able to store discovered NCs in the NCC. Furthermore, it stores topology information in objects. In this object model, its ports and links to other NEs or Internet Access Points characterize each NE. The object model accommodates virtual ports and links and therefore VNEs, see Section 2.3.

2.4.2 The discovery of NEs and Internet access points

The network service is provided through NCs in the application. But how does the application know which NCs are available? Clearly there are many answers to this question. If the NE service interfaces are constructed with web service technologies, the UDDI (Universal Description, Discovery, and Integration) mechanism presents itself. Alternatively, in our experimental implementations NEs are registered manually at the NCC. However, one can do without a central registration of NE service interfaces. This is because that by using neighbor discovery services on NEs, a single NC suffices to discover the services interfaces of all other NEs in a connected graph, e.g. by using `AdjNe` (see section 2.4).

Consider the topology of Figure 10. Suppose the application contains only a reference to NE 5, then the BFS algorithm [74] would find almost all of the NEs. NEs 10 and 11 are not found. NE 10 is connected to 5 only through the Internet and cannot be discovered by AdjNe from 9. NE 11 has neither connection to the Internet, nor to the network of NEs. For applications to interact with NEs like 10 and 11, some other means have to be used to find them.

2.5 UPVN Applications

In UPVN, applications might be distributed, but they will act as a single entity. Once the virtualized network is created, it is available to application developers in the form of NCs, probably implemented as objects. Then, decades of progress in computer sciences can be applied immediately to create new network aware and network centric applications. To illustrate this, we have experimented with implementations of VNS to facilitate an interface with Mathematica [75] on basis of web service technologies. The Virtualized Network Service described in Section 2.4.1 was the result. Figure 14 shows a network topology drawn by Mathematica. Figure 15 and Figure 16 illustrate the apparent ease of the use of VNS by Mathematica. With such an arrangement an enormous wealth of mathematical, computational and visualization software can be applied to solve application specific network issues.

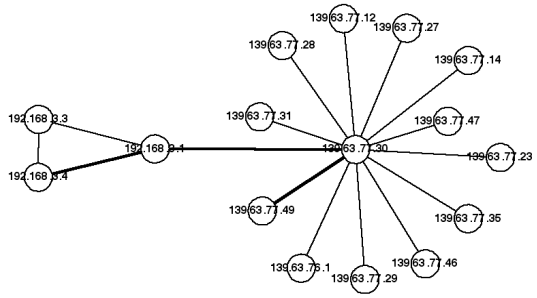


Figure 14. Network topology and visualization of the shortest path (thick line) function in Mathematica's Combinatorica package.

```
Needs["WebServices`"]
<<DiscreteMath`Combinatorica`
<<DiscreteMath`GraphPlot`
Print["The following methods are available from the
NetworkComponent:",InstallService["http://
    localhost:3000/network_service/service.wsdl"]];
The following methods are available from the
NetworkComponent:
{GetAllLinks,GetAllElements,NetworkTokenTransaction}
```

Figure 15. Initializing the network component webservice in Mathematica.

```
n = GetAllElements[];
e = GetAllLinks[];
nids = Apply[Union,e];
Print["Network elements: ", n];
Print["Number of ports found: ", Length[nids]];

Network elements: {bigvirdot,virdot}

Number of ports found: 16
```

Figure 16. NE discovery in Mathematica.

2.5.1 Paths and applications

A whole body of graph theory is at the disposal of the application developer to develop algorithms that satisfy particular purposes. Using flow calculations on basis of actual and near real time data, one can judge how much network capacity is unused. For instance, with a uniform cost search algorithm, it is straightforward to develop an application that finds a least cost route, with a minimum bandwidth guarantee. More sophisticated algorithms allow faster results and faster run time, albeit requiring more understanding from the application developer. Figure 17 shows the determination of a shortest path with Mathematica's ShortestPath function from the Combinatorica package.

Since our NEs support transaction services, one can write applications where the reservation of a path elements, a video movie and a pizza is committed if all of them are available in a given period! Figure 17 shows also a Mathematica code snippet that invokes a transaction to claim all nodes along a shortest path. The transaction is successful if al key provisioned nodes decode the token "Green". One part of this token has been left on the node as a proof of a successful negotiation of AAA matters [70] for specific AC packet services with the owner of the NE.

Such developments bring the powerful facilities of transaction monitors like CICS [76] in the realm of applications that create application specific paths on nation wide network infrastructures for special purposes.

```
nodePath =
ConvertIndicesToNodes[ShortestPath[g,Node2Index[nids,"192
.168.3.4"],Node2Index[nids,"139.63.77.49"]], nids];
Print["Path: ", nodePath];
If[NetworkTokenTransaction[nodePath, "green"]==True,
Print["Committed"], Print["Transaction failed"]];

Path:
{192.168.3.4,192.168.3.1,139.63.77.30,139.63.77.49}

Committed
```

Figure 17. Example of path provisioning in UPVN.

In the case of OSI and IP protocols, algorithms on routers take care of proper forwarding of packets. In case of failures, the routing protocols automatically find alternative routes, if they exist. IP routing is reactive, it responds to a change. In IP networks a routing reconfiguration as a response to a failing router can cost many seconds. For VoIP applications this would result in noticeable effects. In UPVN, applications are aware of the network state and can anticipate problematic situations. ACs in NEs could careful test and monitor the health of a single or a group of NEs (e.g. along a path) to the point that a malfunction is detected before or immediately after it occurs. Applications could then instruct NEs to change their routing tables and their routing policies to forward packets along alternative paths, if they exist, that satisfy application needs. Furthermore, by a modification[2] of the uniform cost algorithm, one can easily find (if they exist) the least cost N alternate path's between a given source and destination, yielding another strategy to find robust path's. In sensor networks the alternate path method not only adds to the reliability of sensor telecommunications, but also distributes the power consumption more equally over the nodes. This is, in many cases, a good strategy to increase the deployment time of the sensor infrastructure that uses batteries.

[2] Find the least cost path by the uniform cost method. Remove all NEs present in the least cost path between source and destination from the search input. Rerun the algorithm to find the next least cost path.

2.5.2 Topology changing applications

In some applications, topology is a concern and application programs have to deal with this. Once some or all NEs are known, one is indulged to make a map of their topology. However, we do not expect programmers to write programs for specific topologies. The application designer mostly does not, and cannot have detailed knowledge of the network. Application programmers rarely construct networks. In the case of wireless sensor networks, optimal application topologies may vary constantly.

In the previous section, we have shown how NEs in a network can be found, how path's can be established in an optimal way and elaborated on strategies by which applications can maintain a certain service level from one or more paths between source and destination. There is another situation, one that we explicitly want to deal with, in which application developers are not aware of the actual network topology. This situation occurs frequently in wireless sensor networks where nodes continuously join and leave the network during application run time. As an example, consider a situation in which the occurrences of articulation vertices, a single node whose failure can isolate part of the network, are unwanted. The Mathematica program listed in Figure 18 finds these vertices with the function `ArticulationVertices` from the Combinatorica package. If sensor NEs would have the ability to create new connections, e.g. by increasing the emitting power of its transmitter or by adjusting the directional sensitivity of its antenna, the NEs could be instructed to change these parameters until Mathematica calculates that articulation vertices have disappeared.

```
ConvertIndicesToNodes[ArticulationVertices[g],nids]

{139.63.77.30,192.168.3.1}
```

Figure 18. Mathematica's detection of articulation vertices.

2.6 Application Configurations

Until now we did not elaborate on where the applications run. A very plausible configuration is one in which the (distributed) application connects via the NCs to the NEs. At this point we say that the application is external to the network.

The NCs may only be available on the NE. Therefore the application itself must then be contained in the NEs. In the special case of a single application, it has to travel from NE to NE to interact with each NC; the application becomes an agent [64]. Traveling here is the process where the NE sends to another NE the application code and the persisted state of the application. VMware has the ability to freeze an entire operating system with running applications and to send the frozen system to another computer. There the execution of operating system and everything it runs is continued. In collaboration with Nortel, we demonstrated this ability at Super Computing 2005 [77] by moving a Linux operating system, running a picture database search application, between computers located in Amsterdam, Chicago and in San Diego.

There are various ways to implement communications between the NC in an application and its NE. Clearly, in some situations one could design a separate network that prevents unwanted interactions of ACs with NC-NE communications. In other cases, such a separate network is not possible and one could use certain ACs to increase the reliability of NC-NE communications. One could for instance design an AC that makes the transmission of information destined to ACs more reliable by implementing a logging and store-and-forward mechanisms. Here one could copy some of the technologies message queuing deploys. With the use of tags containing secured tokens, the NE can discern the appropriate packets in the incoming streams easily. To this effect, NEs may fetch appropriate ACs such as performed in discrete Active Networks [32].

2.7 Conclusion

UPVN shows that well-known concepts for programmable networks augmented with insights from Grid leads to new and practical applications. UPVN applications can deal, by programming the NEs, optimally with changing Internet conditions.

The UPVN concept is foremost useful in situations where structural tuning of applications and network nodes is not feasible any more. Trivially this occurs when many NEs in many network domains are involved. Such situations occur also when networks and applications run close to their maximum capacities and financial budgets – continuous tuning is necessary. Furthermore, we showed UPVN to be equally useful in future mobile sensor network applications, where the communication facilities need to be tuned regularly to the changing state of the sensor nodes.

3

Supporting Communities in Programmable Grid Networks: gTBN

The map is not the territory.

Douglas R. King

Cristea, M.L.; Strijkers, R.J.; Marchal, D.; Gommans, L.; De Laat, C.; Meijer, R.J., "Supporting communities in programmable grid networks: gTBN," *Integrated Network Management, 2009. IM '09. IFIP/IEEE International Symposium on* , vol., no., pp.406,413, 1-5 June 2009.

In this publication, I have contributed to the concept, architecture, and the implementation of token support in the Linux kernel.

This chapter presents the generalized Token Based Networking (gTBN) architecture, which enables dynamic binding of communities and their applications to specialized network services. gTBN uses protocol independent tokens to provide decoupling of authorization from time of usage as well as identification of network traffic. The tokenized traffic allows specialized software components uploaded into network elements to execute services specific to communities. A reference implementation of gTBN over IPv4 is proposed as well as the presentation of our experiments. These experiments include validation tests of our test bed with common grid applications such as GridFTP, OpenMPI, and VLC. In addition, we present a firewalling use case based on gTBN.

3.1 Introduction

Cooperation between organizations, institutes and individuals often means sharing network resources, data processing and data dissemination facilities. Communities are a group of individuals, organizations or institutes that have an agreement about sharing services and facilities, which are accessible only to its members. When user applications need to access and process data from various, possibly heterogeneous, systems and locations, we need to cope with application-specific connectivity, different access policies, and at the same time provide services bound to the community.

The following three scenarios illustrate community-based network services. First, many scientists are allowed to access worldwide digital libraries on behalf of their academic organization regardless of their location (network source address). Currently, this is only possible within the organizational domain. Second, large-scale experiments by scientific communities require data gathering from multiple sources such as high-throughput sensors, lab equipment, followed by data processing on multiple Grids under different ownership. This needs infrastructure support for sharing computational, storage and networking resources. Third, rules and laws of a country or organization may apply to communities of which their members are located worldwide and interconnected over public and private networks and hence, the network has to guarantee separation of these communities to support judicial territories. The three examples all require a form of traffic identification and control to provide specific services. However, the current Internet model offers only best-effort end-to-end connectivity.

In this chapter, we address network support for binding specialized, application-specific services to communities and their applications.

An alternative to the Internet model is programmable networks. In programmable networks, network elements become fully programmable devices. By programming the collection of network elements a network can offer specific services, and implement any form of traffic identification and control. Efforts in programmable networks have led to frameworks, such as integrated and discrete active networks [78], and among others resulted in test beds like Tempest [79], Switchware [80] and Capsules [32]. Although less flexible, optical and hybrid networking technologies (e.g., UCLPv2 [20], GMPLS) are now pre-

ferred over programmable network solutions to provide application controlled end-to-end connectivity. Alternative to current programmable network projects such as GENI [81], OpenFlow [22] offers a pragmatic, intermediate solution based on flows to allow researchers to experiment with new network services and communication protocols, and also have vendor support. While programmable network architectures in general follow a network centric approach, the User programmable Virtualized Networks (UPVN) [82] architectural framework takes an application centric approach by allowing network ser- vices to be defined by distributed and networked applications themselves. The UPVN architectural framework considers the network and its services as software, and defines the elementary components to develop specialized services from applications.

In this thesis, we propose an architecture, generalized Token Based Networking (gTBN), that provides binding between distributed networked applications and programmable network services. gTBN uses the concept of Token Based Network- ing [83] to associate streams with UPVN services by tagging packets with a service identifier. These identifiers are inserted in the process of network resource allocation and management and are independent of communication protocols. We present a reference implementation of the gTBN architecture in IPv4 and a firewall use case as illustration of an alternative approach for domain protection in Grid networks. In a broader context, gTBN allows communities to define their own personal Inter- net providing network services that match their requirements.

The remainder of this chapter is organized as follows. The gTBN architecture is presented in Section 3.2, followed by implementation details in Section 3.3. We evaluate a firewall use case for Grids and show the results of our experiments with streaming, client/server and message passing applications in Section 3.4. In Section 3.5, related work in the field is compared and discussed, followed by the conclusions in the last section.

3.2 gTBN Architecture

The Token Based Networking (TBN) architecture was initially introduced to establish light paths over multiple network domains [83]. Light paths are setup on behalf of authorized applications that need to bypass transit networks. On the one hand, TBN uses a secure signature of pieces of an IP packet as a token that is placed inside the packet to recognize and authenticate traffic. The applications traffic is first tokenized by the TokenBuilder of a local domain (e.g., a campus network), after which it is enforced by the TokenSwitch at each inter-domain controller along the end-to-end path (see Figure 19). On the other hand, TBN makes use of a separate service and control plane. The control plane consists of a AAA server in the push sequence as explained by the Authorization Authentication Accounting (AAA) framework (RFC 2904). The AAA server acts as an authority that is responsible for the reservation and provisioning of the end-to-end paths, possibly spanning multiple network domains.

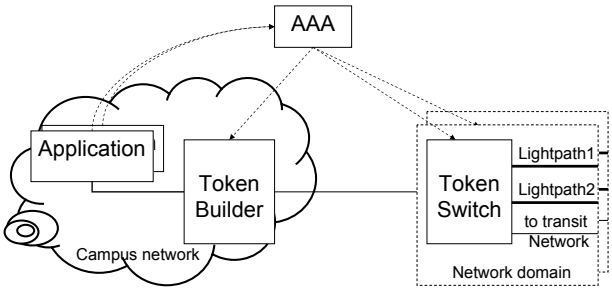


Figure 19. On behalf of an application, the AAA authority provisions the network resources required for an end-to-end light path that may cross multiple domains. At runtime, the application traffic is first tokenized by TokenBuilder and then subsequently enforced by each TokenSwitch along the light path.

Making tokens protocol independent has an important advantage; the token can be regarded as an aggregation identifier to a network service. Generally, we see four types of aggregation identifiers that can be combined, as follows:

- Identifier to point a service to the NE (e.g., a multi-cast, or transcoding),
- Identifier that defines the service consumer (e.g., the grid application),
- Identifier that defines the serviced object (e.g., the network stream),
- Identifier that defines the QoS (security, authorization, robustness, deterministic property, etc.).

First, a token can bind to different semantics and services (e.g., network services for a user, a group of users, or an institute). The semantics that is referred to by a token (e.g., a certain routing behavior) can be hard-coded into a switch or the token can refer to a stored aggregation identifier that points at the specific behavior in a programmable network device. Hence, a token provides a generic way to match applications to their associated network services. Second, tokens can be either embedded in the application generated traffic or encapsulated in protocols where embedding is not supported, such as in public networks. Third, tokens can also provide a general type of trusted and cryptographically protected proof of authorization with flexible usage policies.

The gTBN architecture consists of three parts, as illustrated in Figure 20. First, each community with specific policies and rules needs to be associated with a token (1). A third party implements the negotiation, filtering and reservation of the resources and services using a AAA framework. The implementation of this entity depends on the context, such as resource brokers in Grids or network operators in private networks. Second, specialized network services (e.g., routing, transcoding) for a community are provisioned in the network as required by the applications through their associated tokens. Because the network can-

not know in advance which services or combination of services a community may require, specialized services must be uploaded to a network as ACs (2). Last, when a member of a specific community executes an application, the member or its secure execution environment tokenizes the produced traffic (3). The network recognizes the tokenized traffic and applies the specialized pre-programmed services.

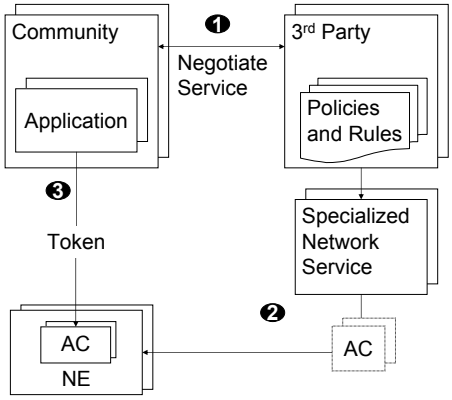


Figure 20. Applications from a community are associated with tokens according to existing policies (1). A third party manages the network resources and provisions the required services into the NEs (2). At runtime, tokenized traffic sent by applications is recognized and authenticated by the network (3).

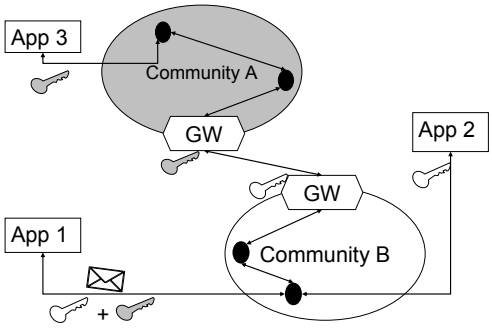


Figure 21. Application 1 can communicate with Application 2 in community B by using the associated token. To communicate with Application 3, the message needs to contain both tokens of communities A and B.

Figure 21 shows an example application of gTBN. Two communities with its member applications are located on different network domains. For example, Application 1 and 2 belong to community B, and Application 3 belongs to community A. The two communities are each associated with a token, gray and white, respectively. Let us assume that

the policy of a network domain is that only members of the communities may be routed through the network. Application 1 and Application 2 are both members of community B and therefore, they can communicate. Correctly tokenized traffic will be routed using default IP mechanisms. However, to communicate with Application 3, Application 1 also needs credentials to access the resources of community A. gTBN supports binding of multiple domain-specific services into a single token. This allows a member of both communities A and B to access each other's network domains.

3.3 Implementation

The complete architecture is composed of three components, as previously presented in Section 3.2 and 20. The first component is a programmable network element that implements the network behavior by recognizing the authenticity of the tokenized traffic. Enforcing the authenticity of tokens ensures the correct execution of the intended network behavior (see Section 3.3.1). The second component runs on the end-user host and binds the token to the application's traffic (see Section 3.3.2). The third component, which is beyond the scope of this thesis, associates a token to the policies applied to the communities; interested readers are referred to [84].

Currently, our implementation binds application's traffic to tokens at the IP layer, and enforces the tokenized traffic at the IP layer, too. It is important to notice that according to the RFC 791 the IP option field, we used for tagging, must be implemented by all IP modules. However, in practice we found that the RFC is not respected by all Internet routers. In a future implementation, though, we will put the tags into an IPv6 extension header.

3.3.1 Programmable Network Element

Our implementation of the network elements (NEs), called Token Based Switch (TBS), must be able to enforce specific network behavior (e.g., routing, multicast) on per-packet basis as required by the tokens the packets carry. Currently, it is possible to implement such programmable NEs using specialized hardware (e.g., network processors, FPGAs) or using powerful PCs. We have chosen to use the network processors because we think that such packet enforcement systems, working at the Ethernet layer, will be located at the gateways of the network domains and hence, they need to process packets at multi-gigabit speeds. For example, Intel IXP2850 network processor provides high-speed packet handling (up to 10Gbps) and on-chip crypto hardware supporting commonly used algorithms: 3DES, AES, SHA-1, HMAC. Although the current powerful PCs are able to route packets at multi-gigabit speeds, they still lack of ability to perform cryptographic algorithms at line rates despite the multi-core architecture.

The current Token Based Switch (TBS) implementation uses the dual network processors hardware platform (see Figure 22). Each NPU contains on-chip 16 multi-threaded RISC μEngines running at 1.4GHz, a fast local memory, registers and two hardware crypto

units for encryption/decryption. The μEngines are highly-specialized processors designed for packet processing, each running independently from the others from private instruction stores of 8K instructions

As illustrated in Figure 22, the Ingress NPU receives incoming packets. These packets can be processed in parallel with the help of the μEngines. The packets are subsequently forwarded to the second NPU. The second NPU can process these packets and then decide which will be forwarded out of the box and which outgoing link will be used.

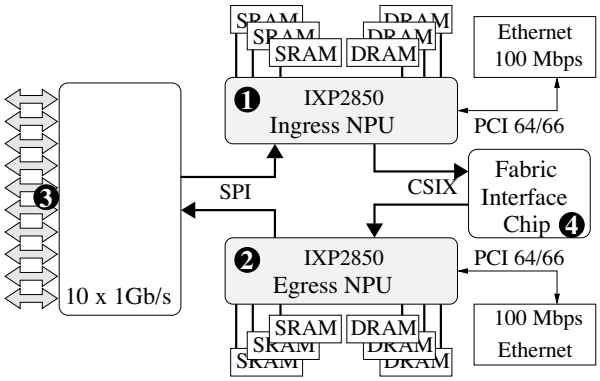


Figure 22. IXDP2850 development platform uses dual IXP2850 NPUs (1) and (2), 10×1 Gbps fiber interfaces (3), a loopback fabric interface (4), and fast data buses (SPI, CSIX). Each NPU has several external memories (SRAM, DRAM) and its own PCI bus for the control plane.

In the current implementation, the TBS uses an authentication application component (AC) combined with a routing network behavior. In other words, the specific routing service run on each authenticated packet. A packet is authenticated when the built-in token (stored in the IPv4 option field) matches the result of applying a keyed Hash Message Authentication Code (HMAC) algorithm over the entire IP packet, or over part of the packet. Our implementation uses the first 64 bytes of the packet to ensure constant speed processing while the HMAC algorithm creates a one-way hash that is a key-dependent (see RFC 2401). In our implementation we opted for a strong proof of authorization by means of HMAC-SHA1 that is also hardware supported by the IXP2850 network processor.

Figure 23 shows the token creation and checking mechanism. For each received packet, the TBS checks whether the current packet has the appropriate IP option field. On success, a Global Resource Identifier (GRI) field, identifying a specific application instance, is extracted (1) to refer to the network behavior needed to be applied to the packet. For example, the expected network behavior is authentication and hence, the GRI points to an authorization table (AuthTable) of an already deployed authentication AC. Next, the authentication AC uses the GRI entry to retrieve the encryption key (TokenKey), already provisioned in the

TBS (2) by a AAA authority. Then, the first 64 bytes of the packet data are masked and then are encrypted using the HMAC-SHA1 with the TokenKey (3). The result is compared with the remaining of the option field. When they match, the authentication AC authorizes the packet to be forwarded to an adequate port. Otherwise, the packet is dropped. Note that although we refer to token as an entire tag built-in the packet, in our implementation, the tag consists of two parts: a plain-text aggregation identifier (GRI) and an encrypted token.

Summarizing, TBS is an implementation of application component (AC) inside programmable NE, which specifically performs packet authentication for access control at multi-gigabit speeds. However, we mention future implementations of other ACs that will perform QoS for the purpose of providing deterministic communication in grid networks (e.g., processing physics experiment data), will assure multi-level security for military networks, will provide robustness in redundant networks.

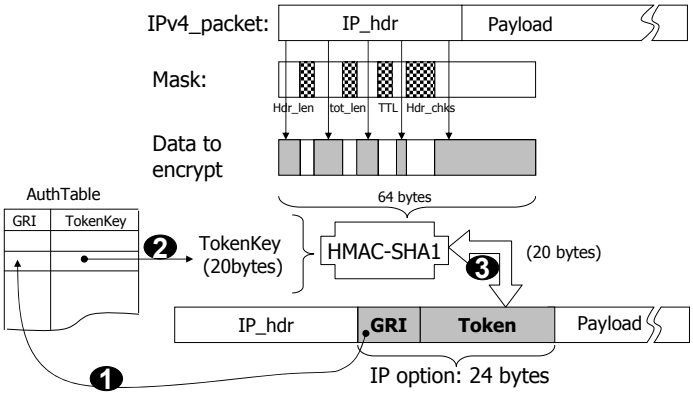


Figure 23. The token checking mechanism uses the GRI part of IPoption tag to point into AuthTable of authentication component (1) in order to extract the TokenKey (2) needed to perform the HMAC-SHA1 over the masked packet data. The packet is authenticated when the encryption result matches the built- in token (3).

3.3.2 Binding token to applications

In our implementation, the binding of token to an application is done on a per-socket basis in order to offer the fine granularity requested by grid applications. Working at socket granularity allows binding distinct streams in the same application to distinct network services. In order to support the token insertion into the optional field of IPv4 packet it is needed to modify the vanilla Linux kernel (1). This modification exposes the token to socket binding mechanism through the `setsockopt()` function of the socket API.

From a practical point of view, it would not be accepted an approach where all the applications need to be modified and recompiled to benefit of a specific token based network

service. In order to provide seamlessness integration of gTBN we developed a middleware using an interposition system such as presented in [85]. An interposition system extends the default socket's function without the need to recompile the application. However, making an unique interposition system working for all imaginable application's scenarios is not possible due to the large variety of socket's usage mechanisms. To overcome this limitation we preferred to support different classes of interposition behavior, called hijackers. These hijackers are installed in the hosts and automatically selected at application start-up via a rule-matching algorithm. The matching algorithm allows users and grid administrators to finely tune how the applications' traffic is tokenized by selecting the proper hijacker. We currently support the following entries in the selection rules: application name, domain, protocol, source/destination IP and port. The rules are checked on a per socket basis; when a rule matches the associated hijacker is started and bound to the socket. We have developed three different hijackers:

- `Hijacker_tokeninjector`, in which each socket that matches the activation rule is bound to a token unique for the application. The token is retrieved from an environment variable.
- `Hijacker_simpletokenizer`, in which each socket that matches the activation rule is bound to a token that is retrieved from a third party supervisor (see Section 3.4.2). The token is unique for a pair of source-IP/destination-IP. This hijacker supports a per-node traffic management.
- `Hijacker_magiccarpet`, in which each socket that matches the activation rule is bound to a token retrieved from a third party supervisor (e.g., a web-server). The token is unique for a pair of source-IP:port/destination-IP:port. Such a hijacker allows differentiating, at the networking level, the different data streams of one application; this behavior is needed in order to support FTP or GridFTP applications that may use distinct parallel socket streams.

The implementations of these three hijackers are used to evaluate several common applications used in grids, as shown in Section 3.4.2.

3.4 Evaluation and Usages

The validation of the approach follows a bottom-up pattern. First we evaluate the performances of Token Based Switch (TBS), our current implementation of gTBN based on the IXP2850 network processor. Then we evaluate the robustness of the middleware by testing commonly used grid applications. In the end we present a use-case in which we applied gTBN to implement a domain firewalling solution for Grids.

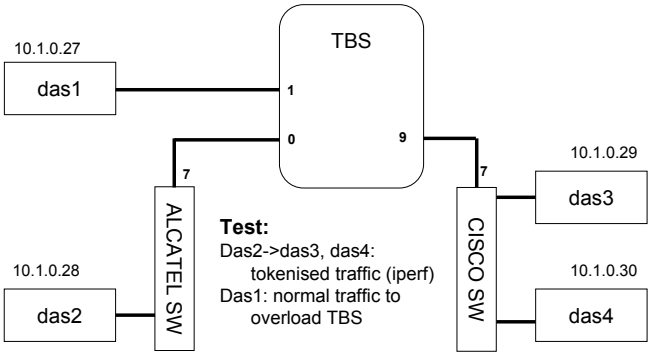


Figure 24. Test bed setup for testing TBS.

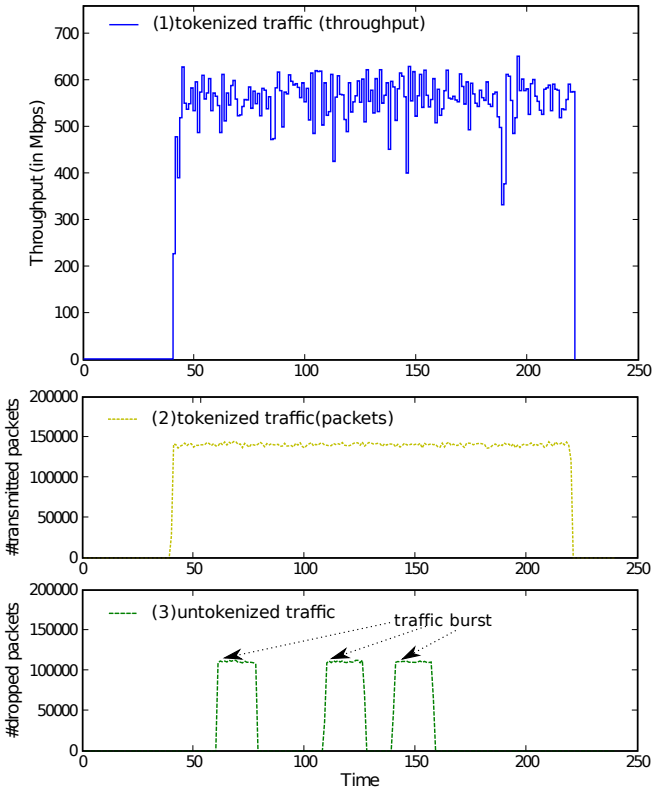


Figure 25. Cross-domain communication between Das2-Das4 begins at 40s and ends at 300s. The traffic is successfully accepted and transmitted. Additionally, un-tokenized traffic is generated between 80s and 190s and dropped.

3.4.1 Token Based Switch benchmark

In order to benchmark our TBS implementation, we built a test bed, as illustrated in Figure 24, composed of four hostPCs (das1 ... das4) interconnected through a TBS. We run the following scenario: das2 send tokenized traffic (generated by Iperf tool as shown in Figure 25 (1) to das3 through the TBS and at the same time, das1 sends traffic to das4, but this traffic is not tokenized and hence, it is rejected by TBS (see Figure 25 (3)).

Such scenario simulates a case when external 'un-authorized' traffic tries to pass or overload a TBS. We measure the effects of such scenario by monitoring the throughput reported by Iperf tool on the tokenized traffic.

As shown in Figure 25 (2), there is no significant influence on the TBS throughput due to the injection of un-authorized traffic. We notice that we can increase the relevance of the evaluation by injecting 'real' traffic such as including random packet sizes, tokenized, un-tokenized, and invalid tokenized. While we could not perform such tests at the moment due to lack of professional traffic generators running at multi- gigabit speeds, in [83] we estimated the outcome by using the Intel's cycle accurate IXP simulator. The bandwidth correctly processed by a TBS implemented on the dual IXDP2850 development platform is around 2.5 Gbps.

3.4.2 Interception middleware robustness

We evaluated the ability of our interposition environment to bind applications to tokens, implemented as described in Section 3.3.2, by investigating the behavior of different applications as regard to their socket behavior. We run the applications over the interposition system and checked with `tcpdump` tool that the applications have their traffic properly tokenized. We repeated the tests on a large panel of real- life classes of applications used in Grid: client-server, message passing, and data streaming, described as follows:

- *client/server:* A server waits for incoming connections and starts a user specific session when a client connects to it. This behavior is typically met in case of file transfer applications. The best way to support these kind of applications is to use the `hijacker_magiccarpet` as this hijacker is the only one capable to associate to each of the connected client a unique token based on the pair of end-point of the data channel. We success fully tested the following FTP servers: muddleFTP and vsFTP with the following clients: netkit-FTP and gFTP. We also successfully tested the grid-ftp application from the Globus toolkit.
- *message passing:* Message passing libraries are important in grids as they provide a widely used distributed programming paradigm for scientific application. We used the `hijacker_tokenin-jector` to bind the token to each of the IPv4 sockets created by the OpenMPI library. By tagging the whole traffic of a distributed application we were able to route the messages through reserved network links, thus providing guarantees on the quality of service.

- *streaming:* The streaming applications, like video streaming or continuously reported data from sensors in live scientific experiment are seldom presented in grids due to the impossibility guarantee on the quality of service of grids network. However, we experienced the VLC real- time video streaming software with the `hijacker_tokeninjector` and Iperf for high ban width data streaming. Iperf was run with '-P' option for the purpose to check the hijacker behavior with sockets used by a typical multi-threaded application.

All of these tests show that it is possible to deploy a working grid environment using a controllable interposition system as a network component (NC) proxy object between applications and network. The impact of this environment on the network performance of the applications is only visible during the socket creation and connection stage.

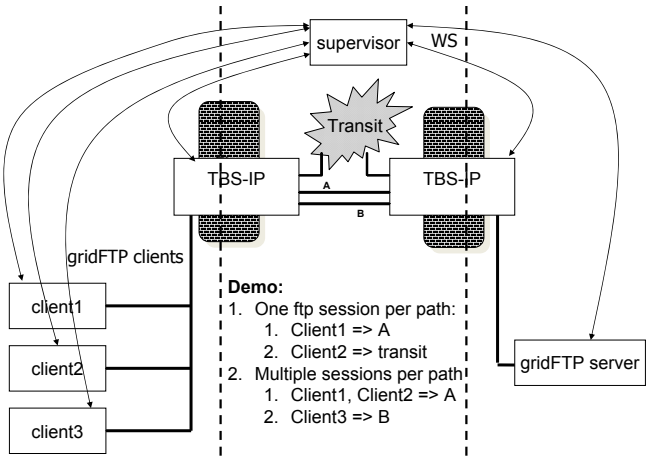


Figure 26. A firewall setup for grids.

3.4.3 Using gTBN for domain firewalling

It is common to have grids interconnected by high-speed backbones, but in many cases they also offer connectivity to Internet. In this case, it is necessary to use a protection mechanism such as Firewalling, VPN, or dedicated connections to isolate and guard the clusters from undesired users.

As a use case, we describe a simpler solution to the existing firewall problem for grid applications (e.g., GridFTP) that open multiple ports dynamically and make difficult or impossible to filter. Our solution based on gTBN is different from existing solutions by not using any protocol related information on the traffic that must pass the firewall, but it rather uses encrypted tokens built-in the packets based on which the packet is authenticated to pass the firewall.

Figure 26 shows the case of two distinct network domains where each is located behind a firewall being interconnected by a public network and dedicated connections (e.g. light paths). Each firewall consists of a TBS being provisioned by an authority: a web-server called supervisor. The test bed also includes four host machines interconnected as follows: three GridFTP clients located in one domain and one GridFTP server placed on the other domain.

The middleware we installed on all hosts uses the `hijacker_magiccarpet` interposition environment in order to tag the traffic of the GridFTP applications. When one host starts a GridFTP client application in order to connect to the GridFTP server, it first gets an authorization ticket from the supervisor. The authorization ticket contains a unique identifier for the requested connection, the so-called Global Resource Identifier (GRI) as illustrated in Figure 23, and a TokenKey needed to encrypt part of each outgoing packet in order to obtain an encrypted token. Second, each outgoing packet that belongs to the socket(s) opened by the GridFTP application gets a tag. The tag is composed of two parts, which are concatenated as follows: (1) GRI and (2) the encrypted token obtained as a cryptographic result performed over the packet as shown in Figure 23. Next, traffic passing the TBSes is checked by looking at the tag each packet carries. If the packet is authenticated to pass the firewall, then it will go out to the provisioned path towards the GridFTP server. A similar scenario of traffic tokenizing happens on the way back from server to client. Note that when the supervisor received a request for path-setup from client to server, it has sent an authorization ticket to all systems involved in the connection: both TBSes and server.

Using this test bed, we investigated the data flow and ensure that the un-tagged packets from one domain are rejected at the entry point of the other domain and hence, only an authorized application correctly bound to a token can enter into a foreign domain. This use case was presented in a live-demo at OGF23, 2008, in Spain as an alternative firewall solution for grids that authenticates the traffic at the granularity of applications regardless of their distributed hosts. To our knowledge, such a fine-grain authentication is not possible using VPNs and is difficult to achieve with authenticating firewalls where the user application must register its flows from all nodes beforehand.

3.5 Related Work

The need for access control, application-specific services, or QoS support for high performance e-science applications has lead to programmable networks and, more recently, to high performance networks with advanced QoS and path configurability. We consider two main topics related to gTBN. First, current strategies and technologies to increase network flexibility within the existing TCP/IP models. Second, the history of programmable networks from active networks to programmability in ATM networks.

3.5.1 Introducing Flexibility in Networks

Currently, there are several network control technologies to help in building network communities at various network communication layers, some of which combining the flexibility of programmable networks with end-to-end provisioning. Furthermore, some recent hybrid solutions also offer cross-layer circuit provisioning. In order to refer the related work in our context, we categorize the existing network technologies with their control features in three classes: (1) embedded, (2) overlay, and (3) hybrid.

The embedded class consists of all network control mechanisms based on embedding specific information (e.g., tags, labels) into the traffic. One example of the embedded approach is Provider Backbone Traffic (PBT). PBT [86] is a VLAN-based solution enhanced to provide a good degree of control over a cluster network as well as on their interconnection backbones. Another example is Multi-protocol Label Switching (MPLS). MPLS was initially developed to speed up switching time by provisioning route decisions in advance, but is now widely used to establish support circuit switching on packet-routed networks, additionally with traffic engineering. It may connect Ethernet or optical circuit-based clients with IP packet- switching clients. Moreover, amongst the latest achievements to support multi-domain networks based on a common policy system, the so-called generalized MPLS (gMPLS) has been developed.

The overlay class includes all technologies that offer network services by encapsulation over existing network technologies, such as IP. Virtual Private Networks (VPN) is a well-known overlay solution to connect private networks or applications through public, untrusted networks. (VPN) based solutions offer authenticated connections over IP. Furthermore, there were past attempts to build dynamic VPNs with fine-grained control down to the low-level network resources, such as the work in [17]. Another overlay network on top of IP, TOR [87], encrypts and obfuscates routing and connections for building an anonymous network.

The hybrid class contains solutions, which combine the existing technologies into one hybrid framework. Hybrid solutions try to achieve the advantages of both circuit switching and packet routing technologies. Examples of such solutions are, UCLPv2 [20], V-STONES [88] and DRAGON [89].

Although the above-classified technologies provide network control and dynamic provisioning of networks, they all limit to end-to-end connections. However, management of dynamic communities is a topic recently tried with MPLS-based VPNs in [90].

In addition to the three classes of control, the middleboxes in a data center (e.g., firewalls, load balancers) offer a different form of control. Firewalls protect private domains from malicious usage by filtering traffic, only allowing specific ports or types of connections. A firewall needs to be well instructed on the traffic it needs to pass or deny. This is specifically noticeable with packet filtering firewall and stateful firewall as they do block/allow access based on specific protocol-related information of the traffic (e.g. port numbers). In order to facilitate the users to bypass such firewalls several traversal techniques are implemented in standard middleware like in [91]. Authenticated firewalls, such as NuFW [92] and

authpf [93], overcome some of the limitations of protocol based firewalls by introducing authentication prior to allowing access. We also notice a recent effort in building of more flexible and easier to deploy middleboxes by using a policy-aware switching layer (PLayer) [94]. PLayer implements the translated policies as specified by an administrator and consists of specific routing tables and switch instructions. Although using PLayer in dedicated pswitches ensures the correctness of a certain possibly complex setup, this approach limits to an existent set of switch capabilities and still uses the complex traffic classification on checking various protocol header fields.

3.5.2 Programmability

Instead of providing flexibility within the context of TCP/IP, another approach is to consider the network itself as programmable. Active networks are the most notable result of the efforts to develop programmable network architectures. The first trials to build active networks date since the ATM age in 90s, when there was a lot of work involved such as the Tempest project [79], Capsules [32], elastic networks [95], SwitchWare [80], xBind. They tried to introduce QoS into ATM networks by enhancing the networks with programmable devices that would process traffic or would self-program based on built-in tags or programs, respectively. Unfortunately, the research on active networks diminished considerably after the year 2000. Around the same time, optical networks gave enough bandwidth and brought different mechanisms for QoS through the end-to-end light paths provisioning. In the end, active networks have not been adopted as a solution to increase flexibility in networks.

In recent years, new developments have led to additional research and new application domains, such as building dynamic communities in networks for the purpose of sharing resources in a secure manner. In the post ATM era, Kindred and Sterne [96] were among the first to describe and address the problem of building dynamically secure network communities over public Internet, though their solution offered a coarse-grain granularity of community members at the level of firewall-protected domains. However, the authors experience shows the difficulty to build an effective community over existing organizations which have different policies and cultures and hence, different ways to distinguish the members from non-members. In other words, communities need a simple way for legacy applications and users to recognize their membership in an existing network infrastructure. Moreover, in the context of grids, a community builds up at the granularity of group members (e.g. user applications or jobs) and eventually different levels of group membership.

3.6. Conclusions and Future Work

The starting point of this work is our believe that in the context of grids, users and communities need the freedom and flexibility to implement their specific network services. To reach this target we introduced the generalized Token Based Networking (gTBN) architec-

ture that combines the programmability of the User Programmable Virtualized Network architectural framework with Token Based Networking. The proposed reference implementation is based on programmable network processors and commodity PCs, and employs an interposition system for seamless integration of gTBN with existing grid applications. A gTBN test bed was set-up and its performance evaluated through a firewalling use case. We also challenged the robustness of the interposition system by executing a set of complex standard Grid applications on our gTBN test bed. These experiments convinced us that multi-gigabit programmable networks are an achievable target for today's Grid networks.

We consider this work as a first step on the road towards a complete Grid implementation of the User Programmable Virtualized Network architectural framework. While still in its infancy, we are now investigating using gTBN for fine- grained access control in dynamically switchable light paths of Starplane, the interconnection network of the distributed computing cluster DAS-3 [97]. Furthermore, we are developing application components that implement more advanced network services like QoS and multi-cast.

Future extensions of this work will include improved resource allocation and brokering, and will enable integration of gTBN into the big picture of Grid's middleware resource and service management, such as VLAM-G [98].

4

Network Resource Control for Grid Workflow Management Systems

One person's architecture is another person's detail.

Robert Spinrad

Strijkers, R.; Cristea, M.; Korkhov, V.; Marchal, D.; Belloum, A; De Laat, C.; Meijer, R., "Network Resource Control for Grid Workflow Management Systems," *Services (SERVICES-1), 2010 6th World Congress on* , vol., no., pp.318,325, 5-10 July 2010.

This chapter presents an architecture and proof of concept to control network resources from a Grid workflow management system and to manage network resources from workflow-enabled applications at run-time based on previous work (Chapter 1). Depending on the network infrastructure capabilities or future advances, applications may employ existing QoS mechanisms or use application-specific ones to provide advanced network services. Our approach leads to performance improvements in communication intensive applications by actively managing traffic flows and enables Grid applications to manage interworking between network and computer resources.

4.1 Introduction

Grid workflow management systems enable smart utilization of computational resources in Grid environments by allowing scientists to plan, schedule and run complex application execution scenarios as part of their scientific experiments. Resource virtualization, i.e. resource as a service, is one of the basic design principles in Grid architecture to make the large amounts of resources manageable and easier to use by scientists. Because most Grid applications have large computational demands, the attention in Grid computing has predominantly been focused on effective and efficient sharing of computational resources.

In recent years, many initiatives have emerged, in which researchers collect enormous amounts of data from the environment, such as dikes [99], the sky [100] or from scientific experiments, such as CERN's LHC detector [101]. By using the Grid, a large amount of resources are at the disposal for such applications, which would otherwise be technically or financially unfeasible to achieve with dedicated systems. The term Sensor Grids [102] loosely defines these types of applications.

Sensor Grid applications are difficult to realize from the network perspective, because they can only execute well, if the underlying network supports their communication demands. On one hand, applications such as e-VLBI, only need high-speed network connections at the time of an experiment, but the required link connectivity may change while the experiment progresses. On the other hand, an early warning system for dike failure might need to redirect sensor data to intermediate nodes in the network for filtering or aggregation before feeding it to computation nodes. Because sensors cannot know in advance to where the data needs to be sent, the network has to be configured on beforehand or adapted at run-time. Unfortunately, such behaviour is hard to achieve in current Grids, because networks do not expose their resources and services to the application domain.

Here, we present the architecture and a proof of concept to control and manage existing network services, such as MPLS [23] or deploy application-specific ones from Grid workflow management systems. The novel idea of our approach is that network elements are virtualized as software objects in the application domain. The application programmer uses the programming interface of the software objects to implement a desired network service. At run-time the workflow management system deploys the application-specific network service on the network elements, which enables applications to control the network elements.

Grid workflow management systems are a natural choice to implement these mechanisms, because they provide abstractions to consider networks as a collection of software objects and already provide similar functions to manage computational and storage resources. Therefore, it is straightforward to reuse and extend existing workflow management system with control over networks. To our knowledge, no architecture or framework is described in literature to control network resources from workflow management systems.

In Section 4.2, we introduce the problem domain, design issues and the basic framework. In Section 4.3, we present a proof of concept using WS-VLAM [98] workflow management system and in Section 4.4 we present preliminary experiments and results to demonstrate the feasibility of network control from workflow management systems. Related work is presented in Section 4.5, followed by discussion and future work in Section 4.6. The chapter concludes with Section 4.7.

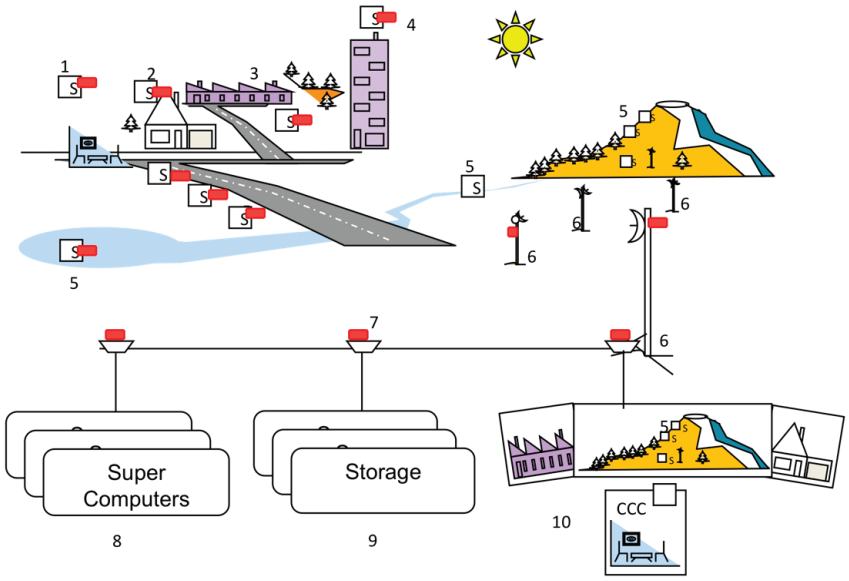


Figure 27. Three major components form a large-scale observation system: sensors (1-5), computers (8-10) and interconnection networks (6, 7).

4.2 Interworking Between Sensors, Networks and Grids

Typically a Sensor Grid application is composed of three main components: (1) sensor networks that monitor environmental properties, (2) computers that process sensor data and (3) an interconnection infrastructure that connects sensors to computational resources, either by using the Internet or dedicated networks.

Depending on type of the observation (Figure 27), sensors (S) need to be distributed at

specific locations (e.g. homes (2), cities (2, 3, 4) or in rural areas (5)). Depending on their situation, sensors are connected via: sensor-to-sensor network, wireless (6) or wired (7) dedicated network or the Internet. When connected to the Internet, applications only get best-effort connectivity. In contrast, dedicated networks can support software plug-ins for application-specific data transformations, such as aggregation, filtering or pre-processing of data streams or conversion of sensor network protocols into Internet or application-specific protocols. Such networks, or the Internet where necessary, connect sensors to supercomputers (8), storage (9) or management systems (10).

Workflow management systems (WMS) provide a transparent and flexible way to compose and execute distributed applications on the Grid. An intuitive approach would be to use a WMS to orchestrate all the components of a Sensor Grid application, i.e. let the WMS care about execution of software components on distributed computational resources and the management of interdependencies and data transfers between these components. But Sensor Grid applications have specific demands that have to be addressed by WMS before an implementation is feasible. The most Sensor Grid demand is to reserve and control network resources to ensure that sufficient network capacity is available to transfer data to computational nodes. WMS already fulfil the task for computational and storage resources. What are the requirements to include network control?

4.2.1 Combined Allocation of Network and Grid Resources

The applications we are considering exhibit a strong sensitivity to their execution time; they are all connected with sensors, record and process real-life data at a given sampling rate. When a sensor produces data, it either has to be stored or processed immediately. Once a radio telescope turns on, for example, it is necessary to receive and process the data as it was transmitted. When this is not fulfilled for a moment, important events may be missed, wrong correlations may be made, and the whole experiment may fail. We consider such applications as *time-critical*. Time-critical applications need to have insurance that the environment is properly dimensioned at run-time. Because Grids can support reservation of resources on beforehand, Grids can provide this insurance. However, in order to support experiments that involve sensor networks, the interconnection networks, shared or dedicated, need to support resource reservation and control too.

In the case where resources are not known on beforehand, for example when an application needs to switch to another data source at run-time, the resource manager has to provide a function to potentially override the reservations made by other applications, i.e. the system has to support rank ordering of the applications in order to decide which applications have priority above others (Figure 28). In addition, the network and computing resources need to be reconfigured on the fly to adapt to the new situation. Because only the application programmer knows how to organize the resources for the application, it needs mechanisms to manage computational resources as well as network resources. Therefore, the resource manager should be accessible through an application-programming interface.

In the Grid the exact locations of the Grid nodes are unknown at reservation time. Therefore, we need to match and reserve network resources after the Grid nodes are allocated. If the network cannot provide the required resources for the application, the computation nodes need to be rescheduled. This process involves coordinated interaction with Grid brokers and network managers and can be dealt with from a WMS. An additional benefit of using a WMS is that it can negotiate and determine the best Grid brokers to submit jobs to, based on statistics gathered from previous submissions. In addition, WMS specialize in dynamic and advanced algorithms for matching, reserving and managing resources.

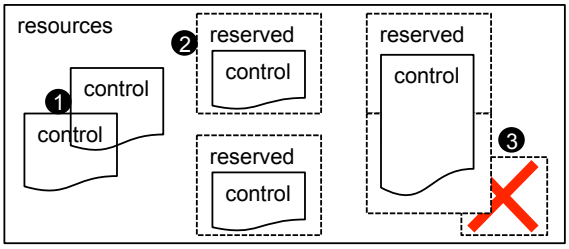


Figure 28. Without reservation network services can conflict (1). Reservation guarantees resources to be available at run-time (2). In order to support dynamic reservation, a resource manager needs to prioritize reservations (3).

4.2.2 The Network as Software Object in Grid Applications

Networks need to provide enough flexibility to support network reservation and management required by our class of Grid applications. In Chapter 2, we presented a concept in which network elements are virtualized as software objects in the application domain. By virtualizing network elements as software objects in the application domain, it becomes possible to control networks using software. Hence, application domain software can be used to program and automate the behaviour and management of network services.

4.3 Implementation

We developed a proof of concept to gain insight in the technical challenges involved in the virtualization of network elements and network control from Grid workflows. For the implementation we reuse and extend existing Grid software, such as the Globus Toolkit 4 [103]. The implementation of the proof of concept consists of two parts, which later are integrated to one solution.

The first part implements an extension of WS-VLAM scientific workflow management system with operations to reserve and control network resources. We integrate a programming interface of the virtualized network elements with the programming interface that WS-VLAM provides to Grid applications to manage computational resources. This adds

network service management to Grid applications, which use the WS-VLAM application-programming interface.

The second part implements a Network Operating System (NOS), which acts as an access point to the network and its resources. In principle, the network management system can implement the task of the NOS. However, network management systems are designed towards the needs of network operators, while we are interested in the challenges to expose the network services to applications. Therefore, the NOS interface (Figure 29, 4) is modelled after Grid broker to facilitate the integration with Grid software.

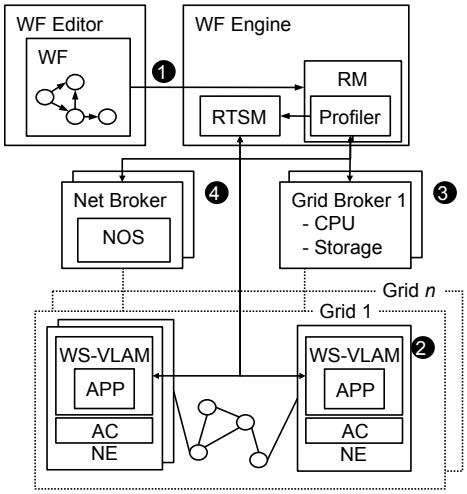


Figure 29. The architecture to control network resources as software objects in WS-VLAM.

4.3.1 WS-VLAM Grid Workflow Management System

WS-VLAM aims to provide and support coordinated execution of distributed Grid-enabled components combined in a workflow. This workflow management system takes advantage of the underlying Grid infrastructure and provides a flexible high-level rapid prototyping environment. WS-VLAM consists of a workflow engine that resides within a Globus Toolkit 4 container on a server side and a workflow editor on a client side. A graphical representation of a workflow is created in the workflow editor (1) and forwarded to the engine to be scheduled and executed on the Grid (Figure 29). The engine consists of two WSRF [104] services: the Resource Manager (RM) and the Run-Time System Manager (RTSM). The Resource Manager is responsible for discovery, selection and location of resources to be used by a submitted workflow. The Run-Time System Manager performs the execution and monitoring of running workflows (2). In WS-VLAM, all distributed applications are represented as a workflow in the form of a data driven Direct Acyclic Graph where each node represents an application or a service on the Grid. The WS-VLAM workflow is data-driven;

workflow components are connected to each other with data pipes in a peer-to-peer manner via input/output ports. When new data arrives to an input port of a workflow component the data are processed by application logic and the result is transferred further via an output port.

Workflow components can be either developed using WS-VLAM library API or can be created by wrapping existing 'legacy' applications into a container. When an application is wrapped the workflow system takes care of state updates and provides an execution environment that allows the manipulation of various aspects of the application.

Workflow components are able to communicate by data streams with each other, with the workflow management system, at run-time with users by means of parameter interface. Each workflow component can have a number of parameters that can be set by a user of workflow management system to control the execution or by the component itself to signal or report some state. For example, the parameter interface can be used by a workflow component to request additional resources like a higher-speed network connection from the workflow management system. The initial WS-VLAM design implied the selection and control of computation resources only – what is typically provided by Grid middleware.

4.3.2 The Network Operating System and Integration of the Network with WS-VLAM

Networks only expose end-to-end transport services, as network details are not relevant to most applications. Exposing control over network services from the application domain enables WMS and applications to develop and manage application-specific services. However, it also imposes complex provisioning issues to the application domain. To address network related provisioning, we introduce a Network Operating System (NOS), which acts as a single point of access to the WMS and hides network-specific complexities, such as provisioning of network elements in a consistent manner. The role of the NOS is to:

1) Manage user/application access to network resources
   (e.g. authentication, allocation, release),

2) Ensure fair usage (e.g. resource budgets, prioritization, scheduling),

3) Prevent errors in network resource allocation
   (e.g. creating paths between not connected nodes, exception handling).

The NOS communicates with clients through dedicated control connections. On start-up, the NOS clients register itself to the NOS. The AC subsystem in NOS clients uses Streamline [36] to load an application-specific network service. We limit ourselves to the manipulation of routes, but Streamline enables the implementation and dynamic reconfiguration of complete routing protocols. The NOS is the entry-point for coordinated access and control of network services. On behalf of the application, it loads or modifies Streamline modules on the NEs. The NOS builds and maintains a network model by executing a discovery mechanism at Ethernet level through which it collects all the neighbours of connected clients.

A chain of packet processing modules (Figure 30) defines which packets are filtered and which the network behaviour is applied. A number of such chains loaded on several NEs define an application-specific network service. In order to provision a network service, which may be as simple as a static path for a source and destination of the application, each chain needs to be provisioned. The NOS uses a distributed transaction monitor to execute the provisioning. When a load or modification of a NE fails, it rolls back the manipulations of all the NEs to keep the network in a consistent state. The NOS assigns each network service a protocol independent token, which is stored in the IPv4 option field. The token is used to authenticate and filter each packet according to the provisioned packet processing chains. This way, tokens associate network services with the traffic in which they are embedded.

A Network Broker (NB) (4) encapsulates the NOS to provide the same function as grid brokers (3), i.e. capabilities to query, request and load network services (Figure 29). Because the resulting interface is similar to Grid brokers, it is straightforward to extend the WMS with an additional service to include discovery, allocation and provisioning of network resources and services via the NB.

```
(netfilter_fetch_in)
   >(fpl_tbs,expression="TOKEN") \
   >(fpl_ipdest,expression="DST_IP")
 >(skb_transmit)
```

Figure 30. A Streamline request in which packets are taken from the Linux Netfilter [37] hook, then filtered by token and the IP destination overwritten.

The flexibility of network connectivity in application and workflow components depends on the usage of the WS-VLAM libraries. If the WS-VLAM API is used, WS-VLAM automatically handles establishment of connections and other technical Grid issues and provides the application developer the resulting data pipes created to its peer (e.g. in the form of C++ or Java IO stream). In WS-VLAM, network connections are implemented with Globus sockets and utilize only TCP (Figure 31). If (legacy) applications are wrapped in WS-VLAM however, connection handling remains the responsibility of the application. Although WS-VLAM does not control such connections directly, it does provide applications the means to control network services. In this case, UDP and other protocols can also be supported.

The network broker receives requests from the WS-VLAM engine via the profiler (see Figure 29). On successful execution of the request, the network broker returns a token to WS-VLAM that associates the network service with the request. WS-VLAM then attaches the token to the data sent by the application. On its turn, the NOS can identify the token (in the IPv4 packet) and apply the associated network service. However, the node on which the application runs has to attach the tokens to the packets. When the API is used, WS-VLAM provides tokenization of application traffic. When the application cannot use the API calls, because the source code cannot be modified for example, WS-VLAM will use socket interposing mechanisms [105] to tokenize the traffic.

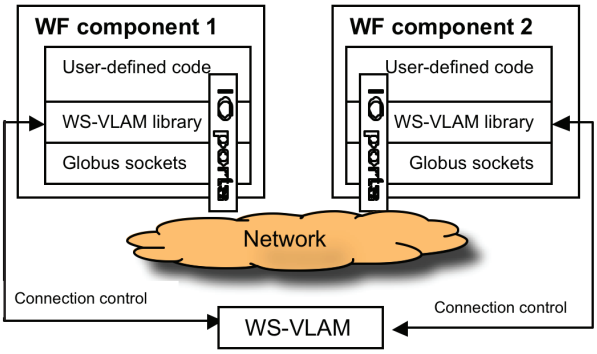


Figure 31. Connection handling for WS-VLAM workflow components.

4.4 Experiments and Results

We developed a test bed and performed a series of experiments to evaluate the proof of concept and the feasibility of our approach. The experiments concentrate on critical aspects of the proof of concept in various application scenarios. We look at scenarios in which the network has to be controlled at run-time, i.e. a continuous loop of monitoring and adaptation to achieve or stay in an optimal configuration. Reservation-time scheduling of network services is trivial; the network is setup just before execution using the same mechanisms. We illustrate the basic steps of resource reservation, allocation and execution of an application in two cases. In one case, the network needs complete reconfiguration, because the data sources change. In the second case, the application requests different connectivity parameters. First, however, we provide a summary of our experimental test bed.

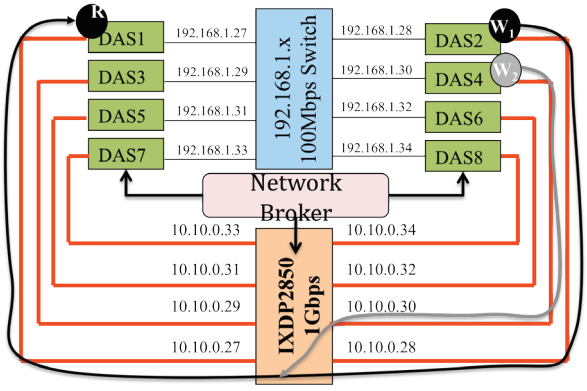


Figure 32. Test bed setup and programmed routes in the first experiment. The network broker accesses the nodes over a separate control plane.

4.4.1 Experimental test bed and performance of programmable nodes

All the nodes in the test bed run Globus Toolkit 4 and also the programmable network software (Streamline) at kernel level. The network broker and WS-VLAM run on separate machines over a control network. Figure 32 shows our test bed in which nodes are interconnected through two networks, as follows: the default network uses a shared 100Mbit switch and the second network uses an IDXP2850 network processor unit programmed to route IP packets at 1Gbps.

Table 1 shows the overhead introduced by our packet manipulation components in Streamline within the kernel of each node. We used Iperf to exchange TCP and UDP traffic between two nodes (over 1Gbps network) in both cases: with and without Streamline. With Streamline, the jitter is larger and the total throughput is lower than without Streamline. However, this overhead is acceptable for our experiments.

Table 1. Packet manipulation performance

	Throughput (Mbps)	Jitter(ms)
Streamline		
TCP	317	
UDP	509	0.026
Non-streamline		
TP	442	
UDP	798	0.009

4.4.2 Run-time request of new data sources scenario

This scenario illustrates how Grid applications can manipulate network services from WS-VLAM. Although changing paths may involve both computational and networking resources, when a new aggregation point needs to be chosen for example, for simplicity we look only into the networking resource brokerage.

Sensor networks and Grids are different systems in reality, but here we assume that a sensor network can provide a Grid service, which can be wrapped as workflow component. We implemented a sensor workflow component in WS-VLAM, which generates random UDP data using `Iperf`. The workflow component is used to simulate a realistic scenario in which the application needs to switch to a different data source, such as a radio telescope, for example, to continue running.

In this experiment, two WS-VLAM workflow components (W1 and W2) are re-directed to a single consumer (R) by an application that processes the data (Figure 32). When the application chooses to request the data of a different sensor, WS-VLAM requests NOS to release the current resources to (W1) and to set up a new path from node (W2) to (R). To make a clear distinction between W1 and W2, IPerf was used to generate data with a bandwidth of 10Mbit for W1 and 30 Mbit for W2, which enabled us to visually verify switching from data sources.

In our current implementation completing a switch from W1 to W2 takes less than three seconds. The mechanisms to load the new behaviour on the network elements, such as a two-phase commit protocol, introduce this delay. In the worst case, a node that does not reply for any reason causes the commit process to wait for the time-out until failure. In the future, we expect to improve the performance of reconfiguring network elements.

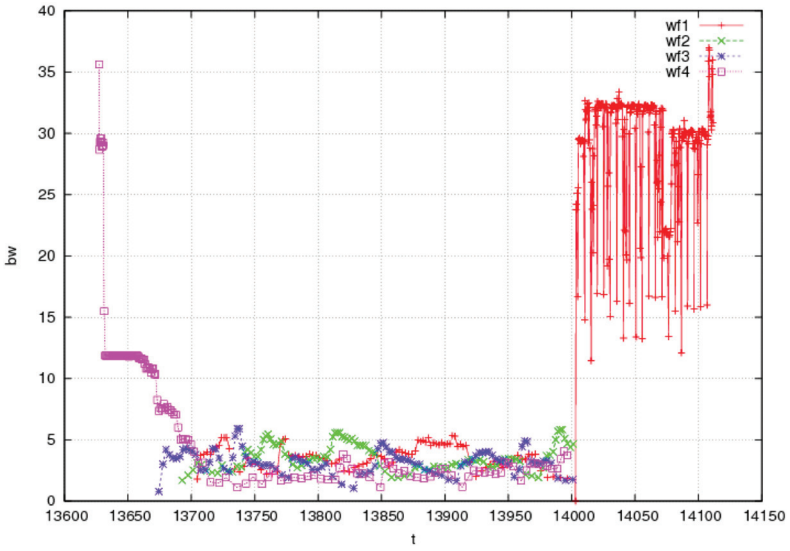


Figure 33. Experimental evaluation of test bed (bw in MB/s and t in seconds).

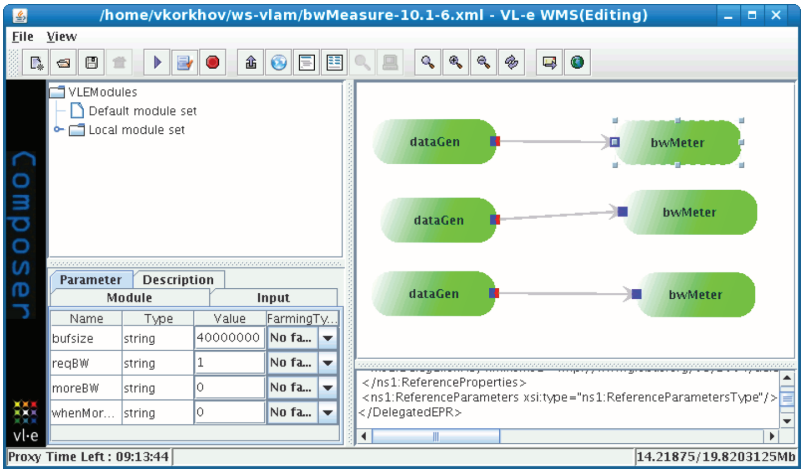


Figure 34. Screenshot of the workflow editor, which shows WS-VLAM workflow where multiple producers (*dataGen* module) and consumers (*bwMeter*) are connected.

4.4.3 Run-time request of better connectivity scenario

Figure 34 shows a screenshot from the workflow manager with multiple workflows. The workflow manager starts workflows one by one: 1, 2, 3, 4. When the network performance (throughput) measured by an application decreases below a certain threshold, the application will request better connectivity from WS-VLAM. WS-VLAM will then offload the resources of the requesting application from the 192.168.1.x network onto the 10.10.0.x network (e.g. path 4 moves to the 1Gbps network), yielding improved performance.

The performance of the experimental application on the test bed is illustrated in Figure 33. Due to the shared 100Mbps, the per-path performance decreases while more paths are established and exchange data traffic at maximum. The switch offers one single network service: best effort. The application running in the workflow (*bwMeter* in Figure 34) measures the throughput and when it reaches a programmable threshold, it requests more resources for the current configuration to WS-VLAM. Next, NOS receives a demand for better paths and decides to create alternative paths over 1Gbps network. Consequently, we see that the throughput increases (*bwMeter* in the second part of Figure 33).

Figure 35 illustrates automatic switching of network paths according to application requests. First, the application is started on a network that provides the throughput of 12 MB/s (section A in Figure 35). Later another application starts using the same network link which results in decreased network performance of the current application (section B). At some point application needs to temporarily increase the throughput and acquire more bandwidth (e.g. for a scheduled bulk data transfer). This happens in the section C: the application switches itself to use another faster network. After the needed data transfer action has been performed the path on the fast network can be released and the initial network is used again (section D).

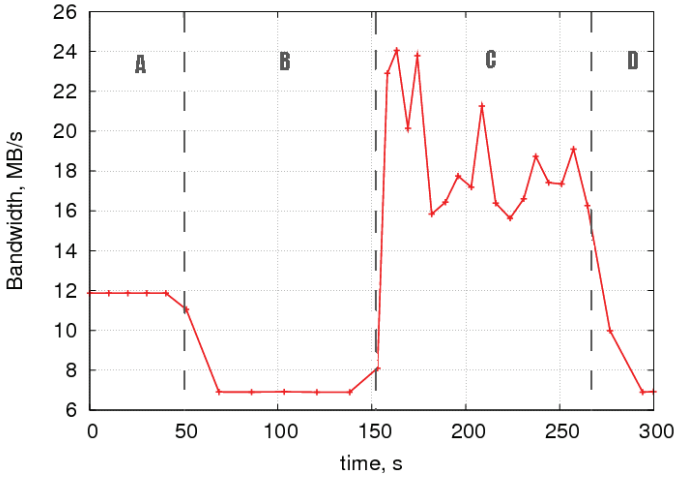


Figure 35. Bandwidth measurements while the application automatically controls throughput.

4.5 Discussion

The experiments with our proof of concept show the feasibility of network control from Grid workflow management system. Traditionally, applications only deal with end-to-end transport services in the network. Our approach changes the way applications can deal with network details. Amongst other things, we need to investigate programming models to develop network services and determine the scalability of our approach. While the proof of concept only supported path manipulation, in the future we plan take full advantage of Streamline to develop complex application-specific routing protocols. Because this is our first attempt, however, many issues still remain.

We experienced the most difficulties in supporting application-specific UDP and TCP traffic manipulation for legacy applications. While supporting UDP is straightforward, supporting TCP is difficult because of the many complex mechanisms that are part of the protocol. Therefore, the behaviour of the protocol has to be understood (and in some cases worked around) while implementing application-specific services.

We noticed that when using the IXDP2850 network processor the jitter increases significantly in the TCP flows. This might be caused by side effects of the code, which processes and rewrites the IP packets. In general, jitter is caused when application-specific code is executed and could be improved taking into account the multi-core architectures. Although decreased throughput is partly caused by the programmable network, a significant performance gain is expected when more efficient mechanisms to capture traffic are implemented.

Our approach to match and schedule the network after the Grid broker allocated the computing nodes sufficed in our proof of concept, but might be inefficient in larger systems. Moreover, when an application requests new computational resources, network resources need to be rescheduled. Better approaches need to be developed and evaluated in real Grid environments.

In this chapter, we have not addressed the issues of managing resources over multiple domains. As with grid brokers, we believe that every network can have its own network broker and NOS. But, to effectively reserve, load and connect network services to applications over multiple network brokers remains a challenge. At least at the network level, tokens in IP packet could be used for multi-domain authentication and for associating traffic to applications. Although technically feasible, the biggest hurdle is expected to be the administrative efforts to allow network resource reservation and control to span over multiple domains.

4.6 Related Work

Coordinated configuration of Grid and network resources is difficult in practice [106]. The effort can be justified in large e-Science applications, however, because it might be easier to run experiments over multiple Grids rather than claiming a large portion of resources in one Grid. But, to achieve acceptable performance over multiple Grids, communication links between nodes the Grids should be optimized. In most cases, this means configuring

dedicated communication paths between the Grids. Progress in grid network research can be characterized by this goal.

Some attempts have been made to incorporate network resource orchestration into workflow management systems. WINNER (Workflow Integrated Network Resource Orchestration) from Nortel Network Labs [107] proposed a way to integrate network resources with WS workflows; DRAC [108] network services are leveraged here for allocation and information in network resource orchestration. However, this project was mostly oriented to business workflows and seems to be not maintained at the moment. In the scope of scientific Grid workflow management systems we are not aware of initiatives to integrate network as a fully featured manageable resource on the workflow and application level.

Several Grid projects attempt to extend control over network resources into the Grid toolset. So far, the efforts focused on reservation, traffic engineering of network circuits and improving the performance of network protocols [20], [25], [109]-[112], but did not expose the new capabilities to workflow management tools where they can be easily accessed and used by applications and end-users.

In our architecture, we have looked at the virtualization of programmable network elements as software objects in workflow management systems. We believe that (1) a workflow management system is the right place to control and manage networks and interface with applications and (2) programmable network technologies are sufficiently well understood to be applied in Grid networks of the future. For example, the next generation of the DAS-4 [113] super computer will include FPGAs in the network fabric. To support our approach in current grid networks, though with the flexibility offered by existing network technologies, we need to take advantage of the progress made in Grid networks. Here, we summarize some of the state-of-the-art Grid networks.

The G-Lambda project successfully conducted the first experiments on coordinated scheduling of network and Grid resources [110], [114]. In G-Lambda's approach, users submit a job to a global Grid resource scheduler, which on its turn allocates resources by negotiating with individual computational and network resource managers. G-Lambda focuses on provisioning optical light-paths between Grids, using MPLS for path provisioning.

UltraLight aims to provide a dynamically configurable network to support high-performance, distributed data-processing application for the high-energy physics community [109]. In their approach, the network is considered a resource and is closely monitored. In addition, the system can take advantage of monitoring information about network state to optimize network resource usage.

The e-Toile project introduces active networks into the Grid domain [115]. By doing so, they go further than end-to-end path reservation and also allow applications to inject code into the network. Their active networks framework TAMANOIR [116] places active nodes at the edges of Grid networks, which can be a platform for adding new and innovative network services.

In non-Grid related network research, other efforts propose to make the management layer programmable [22], [117]. In this approach, computer programs define policies and

network services and control switches and routers that implement flow routing. Such an approach fits well with our goals, because they support application-specific network services and are backwards compatible with current network technologies.

4.7 Conclusion and Future Work

Network performance is crucial to a class of communication intensive applications loosely defined under the term Sensor Grids. Such applications may run in Grids, because it is too costly or unfeasible to develop dedicated systems. Unfortunately, despite the advanced management of computational and storage resources, Grids lack network resource management. To support communication intensive applications or applications with specific network demands in Grids, we introduced a novel architecture. In this architecture, network elements are virtualized as software objects in the application domain and managed by a Grid workflow management system. Moreover, we have built a proof of concept using an existing workflow management system (WS-VLAM) and performed a series of experiments to demonstrate the feasibility of our approach.

Some Sensor Grid applications require immediate access to large amount of resources and control over networks triggered by an event. For example, dangerous water levels detected at dike may result in a large-scale simulation for risk assessment. We have not yet implemented dynamic reservation of shared network resources for urgent computing. Such a feature also needs to be supported by Grid brokers, though WS-VLAM and the network manager can be extended to support application priorities.

While we believe the presented work is a promising approach to support communication intensive applications and to build Sensor Grids, it also raises questions about scalability and security. How can Grid applications control potentially tens of thousands of nodes? Can we use Grid workflow managers to automate parts of the implementation we did by hand? What are the implications for network management and the risks to network operators? In other words, to develop and evaluate practical implementation of the network as a resource in Grids is a topic for further research.

5

Application Framework for Programmable Network Control

With a plan, you get the best
you can imagine.

Adapted from Chuck Palahniuk

With kind permission from Springer Science+Business Media:
R. Strijkers, M. Cristea, C. de Laat, and R. Meijer, "Application framework for programmable network control," *Advances in Network-Embedded Management and Applications*, pp. 37–52, Springer Science+Business Media, LLC 2011.

This chapter presents a framework that enables application developers to create complex and application specific network services based on experience from previous work (Chapter 2-4). The framework expands on the User Programmable Virtualized Networks concept and shows that the typical pattern of an application specific network service is a control loop in which topology, paths, and services are continuously monitored and adjusted to match application specific qualities. We present a platform in which network control applications can be developed and illustrate possible use cases.

5.1 Introduction

Almost every type of network implements measures to guard against unexpected environmental changes, such as the effects of failing links, changing traffic patterns or the failure of network nodes themselves. Such measures can be considered as optimization of network resources with respect to network robustness. At the basis of the optimization of network resources are programs that control the response of the network to changes in and outside of the network. Moreover, actively controlling network resources is crucial to maintain the network service that is delivered to applications.

Optimizations have a certain penalty in realistic situations. For example, in sensor networks [118] minimizing the transmission power of sensor antennae optimizes battery lifetime, but impacts connectivity. Depending on the application and the actual situation, engineers will choose an optimum. Generally, the optimum network service is application-specific, yet in most networks, application programmers have no control over the network. One reason is that a general applicable, conceptual and technical framework to program the network is absent [119].

In the absence of any notion of specific application demands, as is usually the case, network providers can only configure and control the network for best-effort or constant-effort services. Applications can be so specific in their communication requirements and how they tolerate or deal with communication failures. So, theoretically at least, only applications know how to respond to dynamics in computing and network infrastructure. If cloud infrastructures would only run on wind energy, for example, the amount and direction of wind will continuously change the energy available for computing and network resources and their availability to applications. In such cases, (partial) control over the network must also be transferred to a computer program, i.e. the application domain, to facilitate continuous reconfiguration and exploitation of resources.

Computer networks have been designed according to well-defined requirements specified by standards. Application engineers include the network in application logic by using the interfaces of a given network service, e.g. sockets in the Internet. Here, we extend the interfaces to the network such that application-specific network service demands become a network control issue programmed in the application domain, i.e. a dynamic user network interface. We show the consequences of extending the interfaces to the network into the application domain.

In Section 5.2 we review state of the art of related areas in programmable networks, overlay network and sensor networks that allow network control from the application domain. Then, in Section 5.3, the application framework is presented and its functional components are described in Section 5.4. In Section 5.5, the implementation and test bed is introduced and Section 5.6 follows with examples of applications that control networks. The chapter ends with conclusions and future work in Section 5.7.

5.2 Related Work

A basic approach to develop a programmable network is to use general-purpose computers as Network Elements (NE) and implement C programs that manipulate packet streams and network links [36], [56], [57]. The programmable and active network [13], [32] community developed the architectures for dynamic deployment and extensibility of functions in network elements. Other efforts provide programmability in the control plane of networks, while remaining backwards compatible with current Internet technologies [22], [120]-[122]. These technologies enable network operators to offer better services to applications.

Basically, there are two types of limitations in networks that motivate application control: (1) limited network functions or (2) limited network resources. If the network does not offer enough functionality, a well-known approach is to implement the network functions as part of the application, i.e. create and manipulate a virtualized network (overlay network). If the network has limited resources to accommodate application demands in a best-effort manner, frameworks exist to manage the quality of service on behalf of the application [23], [24], [38]. Next, we illustrate some approaches from related network research areas that deal with these limitations.

Overlay networks enable developers to redesign and implement, amongst others, addressing, routing and multicast services optimal to their application domain [123]. Overlay networks are widely used to support specific services, such as distributed hash tables [124], anonymity [87], and message passing [125]. Overlay networks might lead to sub-optimal utilization of network resources, because the mapping to the physical network resources is not open to the application developer. Moreover, overlay networks essentially duplicate functions offered by the physical network. Recently, some efforts [126] propose to expose physical network properties to applications to improve their mapping to the physical network. Assuming that networks are properly dimensioned, at least from the user's perspective, overlay networks are a straightforward solution to support their specific network service requirements.

Sensor networks motivate tight integration of applications and network services [127]. Because of the resource constraints, sensor network designers attempt to use the scarce resources efficiently and various approaches to program sensor networks have been developed [29]. In *Macro-programming* [30], high-level programs use an intermediate language to abstract away concurrency and communication aspects in sensor application programming. A compiler translates the programs into basic instructions for individual nodes, and takes communication characteristics into account. In TinyDB [128], communication is

integrated with a data query mechanism. *Macro-programming* and TinyDB show that with a framework that structures the design space of network control applications, it becomes possible to design and implement reusable components for new applications.

Our research in advanced applications of networks [41], [52], [99], [105], [106], [129] shows that applications have different optimal network services. Existing network management systems do offer APIs to configure network services [130]. Such APIs implement the network abstractions chosen by the network operator. We found that our use cases in hybrid networks and sensor networks require more flexible and specific network services than those designed and implemented by network operators. Because the application domain offers developers more flexibility, it might be more practical to implement network services as part of the application. Hence, we developed a model that enables developers to program networks as part of their application [82]. The resulting framework, User Programmable Virtualized Networks (UPVN), models the interworking between networks and applications and provides a conceptual framework to investigate design patterns of application-specific network services (Chapter 2).

5.3 Application Framework for Network Control

Programmable network element technologies support dynamic network service composition for applications that need new network functions, such as network embedded trans coding of video streams. If changes occur in the network, however, applications must adapt to the new situation. The adaptation process may be at the end-points, such as in TCP flow control process but may also be in the network, such as a process that changes the edge weights of a shortest path routing protocol [131]. The adaptation process typically consists of (1) inferring (possibly incomplete) network information, (2) calculating network state (3) and adjusting the network to a configuration that leads towards the desired optimum. A closed-loop control model, a well-known model in control theory to influence the behavior of a dynamic system [132], provides a minimal framework for network control (Figure 36).

In order to match the network to a state that is optimal to an application, the application has to collect (possibly incomplete) network information. The application developer chooses application specific abstractions (NCx) to update a model the application uses internally. The application combines state information from all or a subset of NEs to update the internal model. In principle, the internal model can also include non-network related information, such as computing or hosting costs, energy usage and service level agreements.

The control application applies an algorithm to find the actions (NCy) needed to adjust the network behavior in such a way that it matches the application needs (e.g. a stable, optimized state), which are described by the reference. To implement changes in the network, the control application translates decisions into instructions, such as create, forward or drop packets specific to each NE involved in the application. This means that the system needs to provide a distributed transaction monitor to keep network manipulations that involve multiple NE consistent.

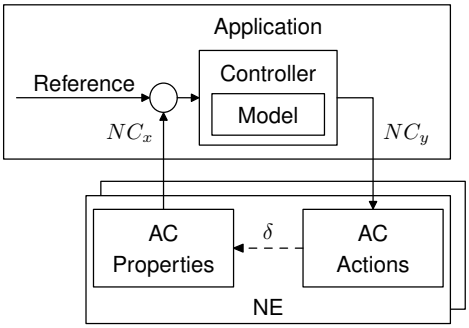


Figure 36. The application framework to control networks contains a control loop.

In control theory, a measurement (AC Properties) from the system is subtracted from a reference value, which leads to an error value as input for the control application. In our framework, the measurements (AC Properties) that represent network state may use different metrics compared to the controlled state (AC Actions). For example, a controller may manipulate edge weights in shortest path routing based on throughput information. Such a scenario is meaningful if the relation between throughput and edge weights (δ) is known or can be learnt and would be useful to dynamically distribute traffic to avoid congestion, for example [131].

Applications exchange information (NCx,y) with NEs over a communication network, possibly over the same network the application is controlling (in-band). Even though application developers may have access to a separate management network, the communication path between network and application complicates the design and validation of the controller. Network properties, such as latency and packet loss, limit the amount of information that can be exchanged or synchronized. So, NE state information can become incomplete, inaccurate or aged. The application developer has to understand the limits in information exchange of a given network, i.e. observability, when designing the control application.

5.4 Functional Components

The OSI reference model organizes the interworking of applications and networks in seven layers [40]. The design principle of layering allows decomposition of a complex problem, but application specific details may be lost in the process. If network elements are virtualized in software, the application interface to the software (NCs) can be fine-tuned to the specific problem domain. However, the fine-tuning might lead to an application specific organization of network functions. Here, we define the organization of functional components to support fine-tuning of the application interface and organization of network functions. The functional organization preserves the context of the NEs. For example, an application might need to manipulate the traffic of a single strategic point in the network for filtering or anomaly detection purposes.

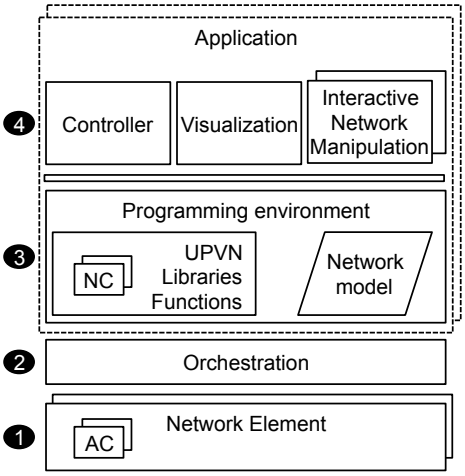


Figure 37. Four functional layers characterize practical application domain network control.

We identify three layers of abstraction in a distributed program: node-level execution environment, interworking framework, and application code. The latter can be subdivided in two sub layers, namely the programming environment providing reusable components such as programming libraries, and the application program. The result is a four-layer architecture (Figure 37). Clearly, the architecture resulting from the application point of view is similar to programmable network architectures [13]. However, the functional components between the application and programmable network need to be further defined to support network control from the application domain and is described next.

The orchestration layer (2) facilitates the interworking of software objects and ACs located on individual NEs (1). The orchestration layer may also support basic mechanisms, such as discovery services, brokers, billing services, authorization, etc. The usefulness of these services depends on the network environment and application. In sensor networks, for example, there just may not be enough computational and storage resources to support an elaborate set of services.

The programming environment, layer (3), provides the NC implementation and reusable components, such as a Distributed Transaction Monitor (DTM) or breadth-first search algorithm, to support programming of a collection of NCs. Depending on the network environment, some abstractions can be implemented in the ACs, as a library in the programming environment or both. For example, the application developer might want to program network element interactions in a non-blocking manner. Hence, either the programming environment or the orchestration layer must facilitate non-blocking interaction mechanisms between ACs and NCs. In our implementation (Section 5.3) we use message passing in the orchestration layer and implemented (an easier to program) blocking interface to the application (Section 5.5).

Because network control is now part of the application domain (layer 4), developers can benefit from a large amount of existing software to implement network control programs.

A characteristic of the control applications is that they operate on data structures that represent the network state. Therefore, the programming environment (3) explicitly contains a model of the network and the orchestration layer must supply the data with which the model can be updated. In Section 6, we discuss issues related to the accuracy of the network model.

Some applications support the construction of a network model that is close to mathematical concepts, such as graphs. The Mathematica [75] environment, for example, contains a graph data structure, which can be used as a basis for control applications that require graph algorithms. By enabling dynamic updates of network state into the Mathematica graph data structure, domain experts can simply apply graph algorithms to find and remove (through network manipulation) articulation vertices; vertices that may disconnect a graph. Besides control, the application layer can also include visualization or other means of interaction with the network. The integration with toolboxes, such as those available in Mathematica, makes the application layer a powerful environment to develop network control applications.

5.5 Implementation and Test Bed

In the preceding sections, we introduced the framework for control applications as well as a four-layered functional model to implement such applications. We developed a test bed according to the presented functional model (Figure 37) to gain practical insight in the implementation of the application framework to support network control programs. The test bed implements the first three functional layers and enables further exploration of the network control applications that are part of the fourth layer.

5.5.1 Hardware

The test bed consists of eight machines (four dual processor AMD Opteron with 16GB RAM and dual port 10Gb NICs and four Sun Fire X4100 with 4GB RAM and 1Gb NICs) interconnected by two 1Gb switches and a Dell hybrid 1/10Gb switch. All machines run VMWare [133] ESXi hypervisor software and the virtual hardware is centrally managed and monitored with VMware vSphere management software. The test bed was bootstrapped with one Linux instance containing the software we developed, and iteratively grown to 20 instances to create a non-trivial configuration of networks and computers (Figure 38).

The setup involves two datacenter locations: a virtual infrastructure running in our datacenter in Groningen and an interactive programming environment including an interface to a multi-touch table running in our lab in Amsterdam. The multi-touch table enables users to interact with NCs (Section 5.6). The two locations are connected by two OSI-Layer 2 Virtual Private Networks (VPN) on basis of OpenVPN [134]; one for control traffic and one for data traffic. At the receiving host in Amsterdam, the control and data networks are separated by VLANs.

5.5.2 Software

The primary purpose of developing a prototype is to gain insight in the challenges and details to control a network from applications that require dynamic traffic manipulation, and to enable experiments with various network control mechanisms. The implementation combines several open source software tools into one NE platform. We provide a global overview of the software that implements the functional layers.

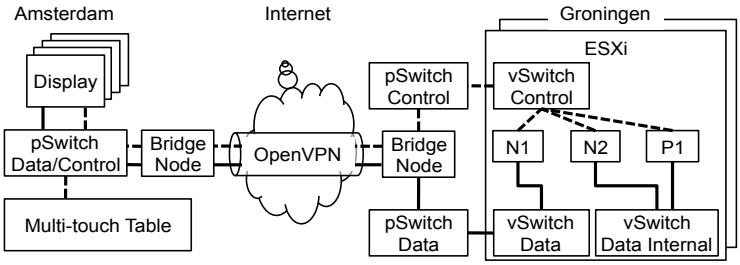


Figure 38. Test bed and network connectivity.

5.5.3 Packet Processing and Token Networking

Fine-grained packet processing and manipulation facilities are implemented in Streamline [36], a tool originally developed for high-speed packet filtering and similar to other approaches presented in literature (Section 5.2). However, Streamline differs from other approaches by providing a simple and flexible query language to manipulate filter graphs on the fly (Figure 30) and a packet processing language FPL [135]. In addition, Streamline also allows dynamic loading of kernel modules that provide specific packet manipulation functions.

We extended Streamline to support insertion, removal and filtering of tags in the IPv4 options field, which allows us to bind ACs to network traffic. A Streamline expression defines a chain of packet processing modules, which describes the network behavior for a particular application on a NE. Filters, such as `fpl_tbs` allow packets with specific tags to pass through a specific chain of packet processing modules. The expression is calculated for each NE separately by the control software and a distributed transaction monitor manages loading of each expression on the subsequent nodes to provision a path, for example.

5.5.4 Orchestration Software

The orchestration of ACs in the programmable network is implemented in Java. ACs available for applications, such as Streamline, are wrapped by Java objects and registered at a local Management agent that knows one or more nodes part of the system. The Management agents provide basic message-passing functions and default socket connectivity. At startup, the local Management agent connects to a known global Management agent.

The global Management agent provides basic services that involve more than one NE, such as a distributed transaction monitor or topology discovery. These basic services are implemented as a set of ACs in the global Management agent and can be used by network control applications. Although the software supports multiple global Management agents, only one instance of the global Management agent is implemented at the moment (Section 5.6).

5.5.5 Network Model

Our implementation provides various active and passive monitoring services bundled in the Monitoring AC that enable network control applications to create and maintain a network model:

- `Ping`: Basic information about latency and jitter,
- Network Mapper (`NMap`) [136]: Detect nodes in the broadcast domain of an interface. For this purpose, `NMap` is configured to send out ARP requests for all possible nodes in the broadcast domain,
- `/proc/dev/net`: Used to retrieve basic throughput information from the Linux kernel. The throughput service returns a summation of the outgoing bytes on all interfaces,
- `Uptime`: CPU load information.

The global Management agent allows ACs to subscribe to events, such as NEs registering to or detaching from the network. The Monitoring AC subscribes to these events and triggers network discovery requests when a NE register, consequently updating its network model to the new network state.

5.5.6 AC Management

Management functions, such as starting, stopping and manipulating AC of the programmable network implementation, are implemented in the Ruby [72] programming language. This allows new network behavior to be added at runtime, e.g. Java classes, kernel modules or installation of complete applications. For example, a ruby script with instructions to compile new code for Streamline and insert it into the kernel can be remotely executed on NEs.

5.6 Network Control Programs

We showed a practical implementation of the model in Section 5.4, which enables a straightforward prototyping of network control programs. To test the setup an interface to view and modify the state of NCs was built. It allows manipulation of video streams

produced by several nodes, which can be displayed on computers with a screen attached. By manipulating the NCs, a user can interact with the programmable network: create and modify paths and modify NE parameters, such as the packet processing chains of Streamline. We successfully demonstrated the setup at Super Computing, 2008, in Austin, Texas [51] (Figure 39).

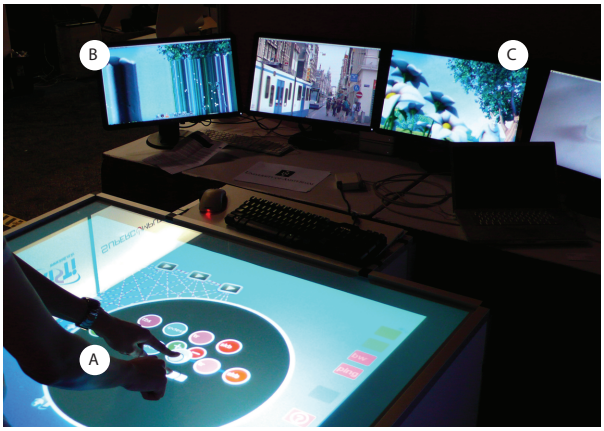


Figure 39. A multi-touch table enables direct manipulation of programmable network components of 20 virtual machines. A user (a) modifies a sampler component of a streamline graph that multicasts a video to screen (b) and (c). As a result, the stream of (b) is distorted, while the other remains normal.

We developed an interactive programming environment with Mathematica, which enabled automation of the possible user manipulations in the setup. Combining Mathematica with programmable networks allows advanced, yet straightforward implementation of network control applications. We implemented a Java adapter between Mathematica and the Management Agent (orchestration layer). The Java adapter deals with limitations of Mathematica's, such as real-time polling of the network, while being responsive to user input at the same time.

The Java adapter enables the Monitor AC to trigger the continuous updates of a number of data structures in the Mathematica kernel, such as `theNetwork` or `thptNetwork`, and facilitates the development of control applications in Mathematica. An elementary control application is one that visualizes the network state while the data structures are updated (Figure 40). For example, the current IP network topology can be displayed while the discovery of the network is in progress; fully discovered in (a) and two intermediate steps (b) and (c). Another visualization example maps throughput measurements on a 3D contour plot (Figure 41). We also implemented various control applications using the test bed. For example, two control applications avoiding congestion were implemented by switching paths and by dropping packets on basis of throughput measurements. Another control ap-

plication was to developed to continuously find and provision disjoint shortest paths [41]. Based on the experiments, we identify new research questions.

Application developers have to consider the accuracy of the network model. For network properties as throughput and delay some range of error can be tolerated. However, applications that require exact shortest paths require accurate topology information. The accuracy of the network model is influenced by the rate at which state information is (1) generated, (2) transported and (3) processed. At least (2) and (3) have architectural consequences for the control loop. One possible architectural consequence is to divide the network in multiple separately controlled domains, similar to areas in OSPF. In one extreme, dividing up the network into smaller individual control domains eventually leads to a fully decentralized architecture, i.e. peer to peer networks. In the other extreme, if network state can be generated, transported to and processed fast enough by one controller, then for practical purposes a centralized implementation might be preferred.

```
In[55]= Dynamic[GraphPlot[theNetwork,VertexLabeling → True, Edgelabeling  True,
  ImsageSize → {630,372}]]
```

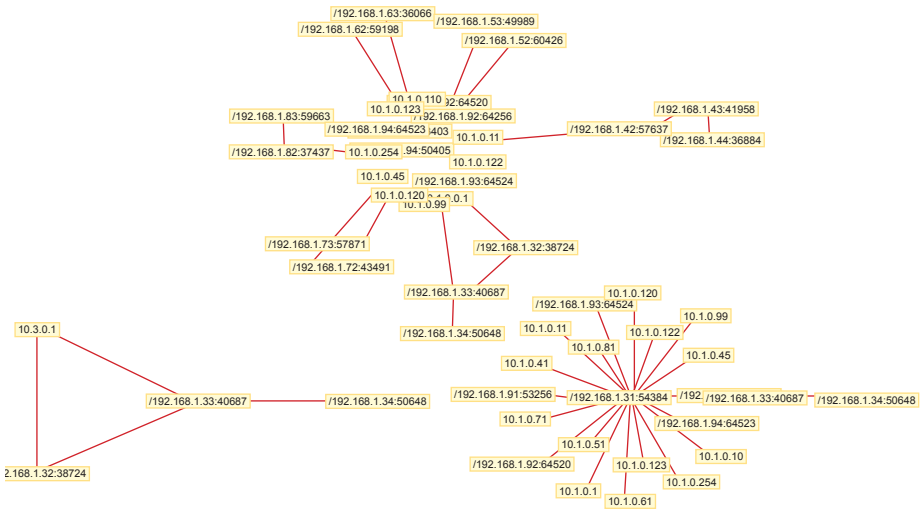


Figure 40. Mathematica's function `Dynamic[]` facilitates continuous reevaluation of network state. The statement redraws the graph every time `theNetwork` data structure is updated with information of the network (a). Picture (b) and (c) show two stages of topology discovery.

Application developers have to make a trade-off between state exchange and the processing capabilities of network elements. For example, an application that finds and removes articulation vertices can run as (1) a centralized component or, in the other extreme, (2) can run on each NE under its control. Because the computation of articulation vertices re-

quires full topology knowledge, running the application on each NE (2) requires additional mechanisms to update and synchronizes changes in topology. Between centralized and decentralized implementations of control loops many architectural variants exist. Likewise, an enormous variety of control algorithms can be expected. On these points applications programmers would benefit from research [137] on design patterns of control loops.

```
Dynamic [ListPlot3D [Append @@ # & /@Transpose[
    {coords, GetVertexWeights[thptNetwork]]],
 Mesh→Full, InterpolationOrder →3,
 ColorFunction → ColorData["SolarColors"], PlotRange → Full]]
```

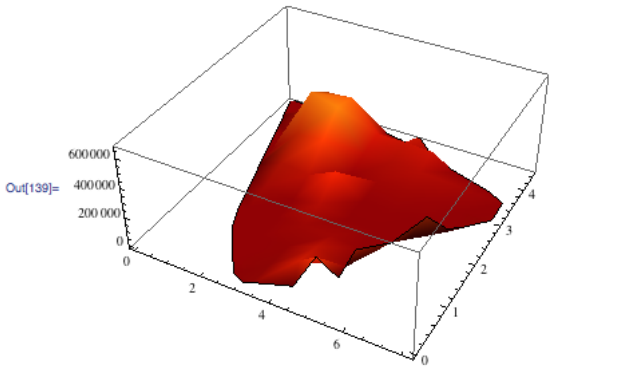


Figure 41. Network throughput of the test bed visualized in Mathematica. The vertex weights in the thptNetwork data structure are updated with throughput values from the programmable network. The network topology is mapped to the x-y plane and throughput to the z-axis (in bytes per second). This way, a user can detect busy spots network and write algorithms to avoid such spots.

5.7 Conclusion and Future Work

Until now, engineers optimize networks at design time and independent of application engineers. Examples from sensor networks, hybrid networks and overlay networks show a need to control networks at run-time. Past efforts created the programmable network element technologies to support dynamic network service composition. In this paper, we use these technologies in a framework for network service development in which each programmable network element has a software representation in a possibly distributed application. We presented an implementation of the framework and several network control applications.

Our implementations are limited to a single application that controls the network. In case many applications want control over the network, another control application is needed to manage (conflicting) resource demands, i.e. an operating system for networks. In the future, however, it can be expected that network management systems support mechanisms to host and run applications on the network. Recent research also continues in this direction (Section 5.2). More experience is needed to create reusable software components that

enable and simplify control application development for large networks.

Control loops are a fundamental part of applications that optimize a specific network service as a response to changes in or outside the network. In subsequent research we shall determine the operational properties of a control application (e.g. how accurate is a given network state, what is the delay between network events and the application's ability to react, how fast can failures be detected). We have shown that architectural consequences can be expected when changes in the network occur faster than a single control loop can effectuate new adjustments, e.g. in large or unstable networks. In this case, the application framework needs to support decentralized network control. Hence, to extend the application framework to support multi-domain, multi-scale network control is a topic for further research.

6

Internet Factories: Creating Application-Specific Networks On-Demand

The Internet is a great way
to get on the net.

Bob Dole, republican presidential candidate

This chapter is based on:
Rudolf Strijkers, Marc X. Makkes, Cees de Laat, Robert Meijer, Internet factories: Creating application-specific networks on-demand, Computer Networks, Volume 68, 5 August 2014, Pages 187-198, ISSN 1389-1286.

We introduce the concept of Internet Factories. Internet factories structure the task of creating and managing application-specific overlay networks using infrastructure-as-a-service clouds. We describe the Internet Factory architecture and report on a proof of concept with three examples that progressively illustrate its working. In one of these examples, we demonstrate the creation of a 163-node IPv6 network over 18 cloud locations around the world. Internet factories include the use of libraries that capture years of experience and knowledge in network and systems engineering. Consequently, Internet factories solve the problem of creating and managing on-demand application-specific overlay networks without exposing all their intricacies to the application developer.

6.1 Introduction

Infrastructure-as-a-service clouds [1] have the potential to become an elementary part of large and complex systems, such as cyber physical systems [2] and ICT systems to support smart cities [3], [4]. Nowadays, we all know Internet applications and services such as Spotify [5] and Youtube [6], which have resulted in dedicated overlay infrastructures (servers, networks, storage facilities) on top of the Internet. Instead of creating dedicated overlay infrastructures for large and complex systems, they can now be implemented entirely in the application domain with infrastructure-as-a-service clouds. Consequently, developers of Internet applications and services are confronted with an enormous increase in complexity. They must deal with network domain complexity and life cycle management of the overlay infrastructure, in addition to the application domain complexity. To make this approach feasible, we need to develop a software architecture in which domain experts - application developers and network engineers - can apply and reuse knowledge in the design, implementation, and management of overlay infrastructures. Consequently, a new paradigm to create large and complex overlay infrastructures emerges in which developers build on experience and achievements of domain experts.

In essence, overlay infrastructures on top of the Internet add capabilities for controlling software and resources at specific locations. A generic way to provide these capabilities is to use virtual machines. Not surprisingly, network research test beds use virtual machines to experiment with new networking and distributed computing technologies [7]-[10]. Even the telecom industry is adopting the use of virtual ma-chines for encapsulating network element functions (such as firewalls, residential gateways) [11]. Still, the emergence of Infrastructure-as-a-service clouds marks a significant step from state of the art, because developers gain programmatic control over the overlay infrastructure and its application services. In the end, application developers only want to deal with the infrastructure details relevant to their application. So, the problem is how to allow selection of certain infrastructure details that a developer needs to cope when designing specific applications. We introduce the Internet factory concept as a solution to this problem.

The Internet factories concept is based on the broader concept of software factories in software engineering [12]. The rationale of software factories is that the majority of appli-

cation development can be captured by standardized components and common patterns. Instead of reinventing the wheel for problems they encounter, developers re-use existing components and customize, adapt, and extend them for their needs.

An Internet factory is a software architecture in which domain experts implement libraries and design patterns to simplify the development of applications that interwork with an overlay infrastructure. It builds on the User Programmable Virtualized Network (UPVN) architectural framework, which models the general programming problem for interworking between application domain programs and the network domain [13], [14]. In this chapter, we refer to these application domain programs as Netapps – Networked Applications, to distinguish from other application programs that do not inter-work with the network domain. So, an Internet factory provides the libraries and common functions with which developers can implement Netapps, such as a cyber physical system, new networking concept [15], or application-specific overlay [16].

The contribution of this chapter is the design and implementation of an Internet factory proof of concept to support the development of Netapps. We developed three Netapps that progressively illustrate the deployment stages in the life cycle of an overlay infrastructure: creation, configuration, and management. The proof of concept exploits resources of infrastructure-as-a-service cloud (from now on referred to simply as clouds) providers whose interfaces are accessible via the Internet. We also show how libraries, tools, and frameworks can be applied to simplify Netapp development. For instance, we describe a library in which the creation of an IPv6 overlay infrastructure is fully automated and how third party applications can be used to adapt networks. In fact, the presented work shows the potential of Internet factories to create large and complex overlay infrastructures.

Related work and the Internet factories concept are presented in respectively Section 6.2 and 6.3. In Section 6.4, the design and implementation of an Internet factory, called Sarastro, is presented. Sarastro is evaluated by illustrating three typical Internet factory patterns in Section 6.5. The implications of Internet factories are discussed in Section 6.6 and the paper ends with the conclusion and future work in Section 6.7.

6.2 Related Work

In the past, programmable networking concepts have been proposed to give application developers more control over the network domain. In its simplest form, a programmable network consists of a collection of programmable network elements, such as commodity PCs, which can simply be loaded with software providing new functionality for processing network traffic. The basic concepts to program a collection of programmable network elements have been developed and validated in active network research [17] and in subsequent research [9], [18]-[20] of which Software Defined Networks (SDN) is currently the most popular [21]. The concepts allow a programmable network to become an active component of distributed applications by exposing APIs to such applications or by allowing applications to embed network instructions in the packets they send over the network. Net-

work operators use the same interfaces to create network services for specific applications or to optimize network services for their own operations. In practice, this leads to an increase in complexity for both network operators and application developers.

For example, application developers typically use sockets to implement communication over the Internet without knowing details of intermediate networks. When using programmable network interfaces, they must also deal with details of the networks, such as routing and topology. Choices about where to place application code in a network, filtering and redirecting traffic to the code, and interventions on failures all require in-depth networking expertise. For any service that reaches be-yond a single ISP, the complexity to setup and manage a network service over multiple domains quickly increases.

Network engineers who understand the network requirements of application developers can implement a solution in the network on their behalf. Then the roles are turned; the network engineer must understand the intricacies of the application domain and anticipate the network functions and services it requires at what moment. With traffic engineering and control plane programmability [21], [22], solutions can be implemented strictly in the network domain. In practice, due to the scale, complexity, and diversity of the application domain, the best that network engineers can do is to over provision and prioritize certain traffic classes.

The only alternative is that application developers create their own services and interventions as an overlay on top of the Internet. On one hand, overlay networks such as Planetlab [7] and content delivery networks [23] both exploit the Internet and implement solutions to reach past its limitations. For example, Planetlab and similar initiatives [8], [9], [24] allow researchers to develop and test new network and distributed computing services, which cannot be done on the Internet. On the other hand, the creation of an overlay networks results in an additional, often dedicated, infrastructure layer on top of the Internet. Thus, developers of these overlay infrastructures require expertise of both the application and network domain and need to deal with their interworking. Internet factories help in compartmentalizing the complexity of common patterns, so the developer can focus on the application-specific issues.

Two additional research domains are related: peer-to-peer networks [25] and autonomic networks [26]. In contrast with the overlay networks previously mentioned, peer-to-peer networks exist exclusively on end-user hosts. Consequently, peer-to-peer technologies focus on delivering a service despite intermittent connectivity, for example, due to hosts joining and leaving the network. Autonomic networks use expert systems to enable unsupervised operation. The complexity is hidden from the operator by algorithms that automate network management procedures. In this work, network management procedures are automated as well. Still, peer-to-peer and autonomic networks address only part of a more general programming problem in the design and implementation of over-lay infrastructures, which is further described in the next section.

6.3 Internet Factories

The general programming problem of overlay infrastructures is described next in the form of the UPVN architectural framework. Then, the typical pattern of creating and managing and overlay infrastructure is presented. Finally, the Internet factory concept is introduced, which provides a solution to simplify the programming of overlay infrastructures in the application domain.

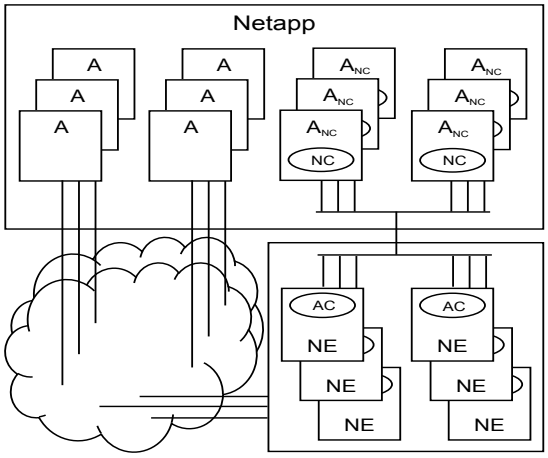


Figure 42. In the User Programmable Virtualized Networks architectural framework, Network Element (NE) functions (AC) are exploited via software objects (A and ANC) in Netapps. Their interfaces determine the amount of control Netapps have over network elements.

6.3.1 UPVN Architectural Framework

The design concept of the User Programmable Virtualized Network (UPVN) architectural framework is to model network elements as a collection of software objects in the application domain [82]. These objects are a manifestation of the network in application domain programs (Netapps), which enables developers to abstract, extend, and integrate network element services in their applications.

For any meaningful behavior of an operational UPVN, software ojects created and manipulated by the Netapp must be bound to the network elements providing the implementation of the software objects in the network domain. The UPVN architectural framework leaves this process to implementations of the framework. To understand why, the architectural framework is shown Figure 42. Netapps create a collection of software objects, represented as 'A's. The amount of knowledge and control a Netapp has over the network is determined by 'A's Network Components (NCs). So, 'A's without NCs provide less detail and higher abstraction of the network than A_{NC}s. Each NC has one or more corresponding Application Components (ACs) implementing the mani-

festation of the 'A's in the Netapp. One may run the ACs in programmable Network Elements (NEs) such as in the kernel of an operating system, SDN controllers, VMs, or future programmable NEs. Depending on the UPVN implementation, ACs can provide specific functions, abstractions, or network contexts. In one UPVN, the binding might be implemented with network management interfaces, such as NETCONF [138], to access physical routers at specific geographical locations. In another UPVN, discovery services or complete programmable network stacks might be used. So, any architectural choice on the binding would restrict the design space of UPVNs.

A straightforward implementation of a single UPVN follows that of a programmable network. For example, a network consisting of commodity PCs could be provisioned with ACs that manipulate packets from the Linux kernel. Consequently, network elements implemented in this way expose (parts of) the network to the application domain, which is shown in Figure 41. Initial UVPN prototypes were implemented this way [28]-[30]. Therefore, they also share the limitations of programmable networks. Cloud services offer simple interfaces via the Internet to create and manage Virtual Machines (VMs). ACs can also be implemented by using VMs. Then, the cloud services interfaces can be used to create and manage ACs. An implementation of the UPVN architectural framework using these interfaces allows Netapps to integrate the life cycle of UPVNs.

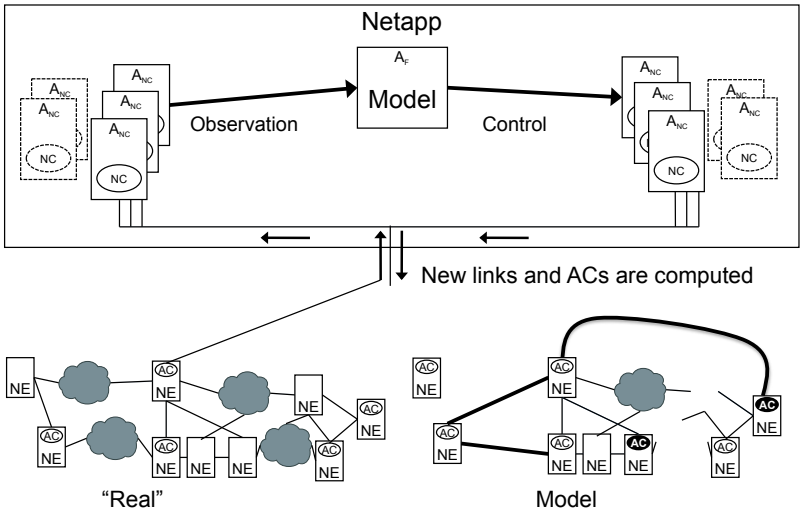


Figure 43. Netapps describe the actions and their effects to bring a user programmable virtualized network towards an optimum. In this research, actions also include infrastructural changes, such as creating a new Application Components (ACs) at specific locations in the Internet.

6.3.2 UPVN Life Cycle

In previous work [139], we showed that a typical design pattern of UPVNs is a control loop in which topology, paths, and services are continuously monitored and adjusted to match application specific qualities shown in Figure 43. Here, the control loop also includes the life cycle of a UPVN. The procedure is similar to the *network management cycle* [140] of any network. In the network management cycle, network engineers and domain experts design and implement a (overlay) network, e.g. network topology, capacity, and configuration of routing protocols. In the implementation of a UPVN life cycle, application developers face the same issues as in the network management cycle, but as a programming problem.

As part of the programming problem, developers not only have to implement interworking between applications and networks, but also the organization the UPVN infrastructure. In the extreme, the UPVN design pattern includes all aspects of a network's life cycle and operation: resource planning, performance and fault monitoring, network and application service configuration, accounting, and security management to name a few. The UPVN architectural framework does not provide mechanisms to hide such details from the developer, as their implementation is application-specific. The purpose of an Internet factory therefore is to provide a structure that simplifies the task of creating and managing UPVNs.

6.3.3 The Internet Factory Concept

An Internet factory is an implementation of the UPVN architectural framework that simplifies the development of Netapps to create and manage UPVNs. As the equivalent pattern in software engineering, an Internet factory encapsulates the complexity of creating UPVNs. 44 shows that an Internet factory serves as an intermediate between Netapps and UPVNs. Its primary purpose is to hide complexity from Netapps, which can be done in two ways:

(1) Directly via the implementation of common patterns, libraries, and domain specific languages that developers include or use for programming Netapps,

(2) Indirectly, via a compiler for example, in which the Internet factory processes the Netapp and creates a control program containing all the procedures to create and manage a UPVN.

In both cases, the Internet factory simplifies the process of creating and managing UPVNs using pre-defined procedures. For example, network engineers could provide a pre-defined procedure to create and configure an IPv6 network (see Section 6.4.4). In the first way, Netapps can use specialized applications to implement application-specific interventions (see 6.5.3). In the second way, the Netapp serves as a blueprint and the control program acts as an executable for creating and managing UPVN (see Section 6.5.2 for a simplified example). As any executable, the control program can be parameterized, copied, and distributed. Control programs can also be part of the collection of pre-defined procedures in an Internet factory. The simplest control program encapsulates an instance of the Netapp

and Internet factory. For example, a Netapp can offer an interface to create a UPVN according to a template (see Section 6.5.2). So, Internet factories allow further compartmentalization of complexity through recursion. Via these constructs, an Internet factory facilitates developers in building complex solutions that build on solutions previously implemented by domain experts.

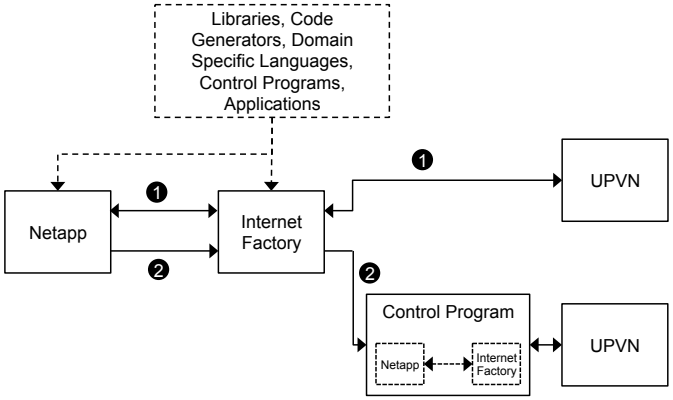


Figure 44. An Internet factory hides the complexity to create and manage a User Programmable Virtualized Network (UPVN). An Internet factory acts (1) directly or (2) indirectly via a control program. The difference is that an Internet factory can further compartmentalize complexity by nesting other Netapps and Internet factories.

Once created by the Internet factory, UPVNs become objects with their own state and interfaces (hence the arrows from the UPVN back to the Netapp in Figure 44 (1)). A task common to all Netapps is the management of nodes and links in the life cycle of their respective UPVNs. As an intermediary between Netapps and UPVNs, Internet factories can encapsulate such common tasks. In our implementation, UPVNs inherit life cycle management functions from the Internet factory (UPVN collection in Figure 45), as it simplifies both the development of Netapps and UPVNs. Therefore, our Internet factory implementation can also be regarded as a platform on which the Netapps execute (see Section 6.6).

6.3.4 Interoperability with Network and Cloud Management Systems

To a certain degree, UPVNs can be created and managed using cloud service interfaces only, though clouds do not yet offer the quality of service that dedicated infrastructures offer. In principle, tunneling technologies are sufficient to facilitate any type of virtual link between ACs running on one or more clouds. For example, a Netapp could use Ethernet VPNs to create an IPv6 network over IPv4 Internet. Clearly, such a UPVN would not benefit from possible optimizations with MPLS [90], SDN, or other methods operators might provide to improve performance of connections and virtual machines.

In general, Netapps require interfaces to query, select, and allocate resources from mul-

tiple clouds, including their interconnection. Intercloud research in part addresses the research problems associated with the diversity in interfaces, policies, and capabilities of using multiple clouds [141]. As Netapps depend on the Internet factory to provide the appropriate interfaces, additional libraries that exploit such optimizations may be implemented. Those additional libraries may also include other resources, such as storage or visualization facilities. In our proof of concept, however, an *information service* (Section 6.4.1) provides only the elementary functions to create and manage UPVNs over multiple clouds. Recent work shows how with control over AC placement, we can avoid 'bad spots' in the Internet, such as routes with unexpected high latency or congestion [142]. An information service might provide the intelligence to include such considerations.

6.4 Design and Implementation of an Internet Factory

In the UPVN architectural framework Netapps implement the design choices and architecture of programmable networks. Of course, the idea of an Internet factory is to provide common patterns to simplify development of Netapps based on these design choices. The proof of concept shows a basic architecture of an Internet factory on which libraries can be built encapsulating these common patterns.

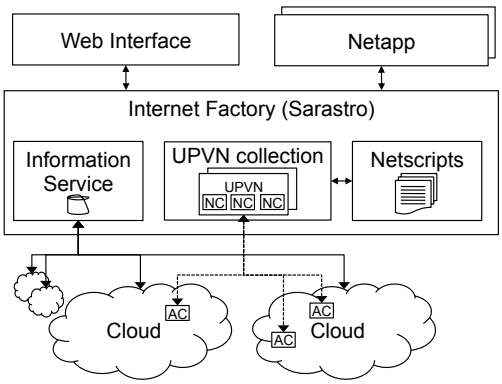


Figure 45. Sarastro is a web application that implements an Internet factory for infrastructure-as-a-service clouds.

The architecture of our proof of concept, called Sarastro, is shown in Figure 45. Sarastro was implemented as a web application in Ruby [143] using the Sinatra [144] web framework, Redis [145] storage back-end, Fog [146] cloud services library, and EventMachine [147] for a-synchronous, event-based data processing. It consists of the following components:

- An Information Service, which provides an interface to query, select, and allocate ACs (VMs) on clouds known to it,

- A UVPN collection, which manages the life cycle of UPVNs and implements a basic interface for node and link management common to all UPVNs,
- And Netscripts, which encapsulate the deployment, configuration, and management complexity of network and application services.

Sarastro exposes its interfaces to a Web interface and Netapps via a Web API (Table II shows the main functions). The Web API is designed according to *representational state transfer* (REST), in which resources are referenced by URLs and operations on those resources are defined as GET (retrieve representation of resource), PUT (manipulate state of resource), POST, and DELETE (create and delete resource) actions. Representational state transfer also allows a flexible and straightforward implementation of the interaction between NCs and ACs in a Web interface or programming language, including unexpected ones (see Section 6.5.3). The Web interface is a run-time environment in which developers can create and manage UPVNs, and inspect their state (see Figure 46). Amongst other state, UPVN topology, properties of known cloud locations, and the state of Netscripts can be inspected and manipulated.

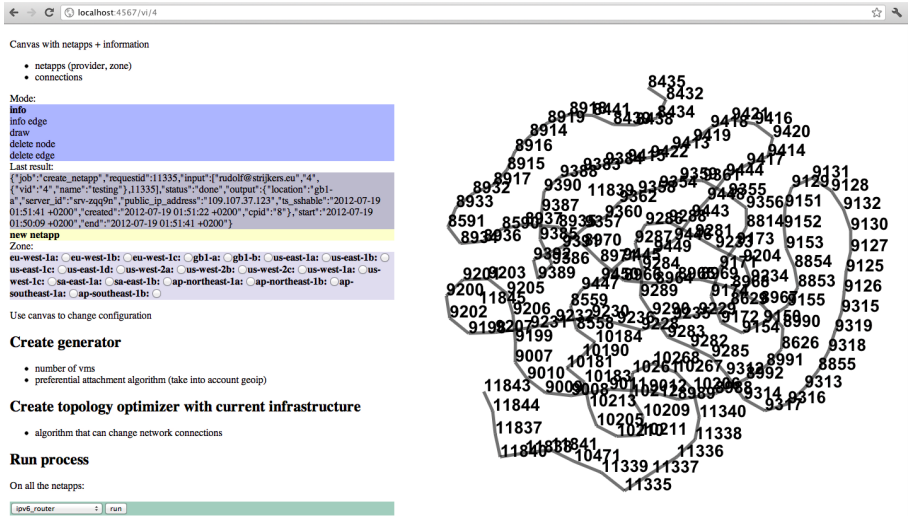


Figure 46. A screenshot of a 163-node UPVN in Sarastro's web interface.

The Netapps for our experiments were predominantly written in the Javascript programming language, which can execute from a web browser. Nowadays, web browsers that support HTML5 implement advanced technologies, such as event sources and asynchronous code execution. As Sarastro is implemented as web application with a REST interface, Javascript is a suitable language for rapid prototyping of Netapps, including visualization and end-user interaction.

Sarastro stores credentials for all cloud service providers it may access. Therefore, develop-

ers must be authenticated to use the Web interface and to make API calls with Netapps. Authentication is implemented with OpenID [148] against Google accounts. Once authenticated, developers can access cloud credentials and those of all running UPVNs, including the SSH certificates to log in to ACs. Next, the implementation of Sarastro's components is further described.

Table II. Packet manipulation performance

REST Interface	Description
`GET / feed`	Subscribe to event of a UPVN
`POST / event`	Post events to listeners
`GET, POST / node`	Manage Acs
`DELETE / nod/id/:id`	
`POST /node/:nid/run`	Run a netscrript on one AC or
`post /upvn/:vid/run`	all the ACs in the UPVN
`GET /node/links`	link management
`POST /node/link`	
`DELETE /node/link/id/:`	
`GET /upvn/:id`	Grouping of links and nodes
`Post /upvn`	into UPVNs
`GET, POST /cp`	Cloudservice provider
`DELETE /cp/:id`	management

6.4.1 Information Service

Netapps control the allocation and placement of ACs with the information service. The information service implements elementary functions to query, select, and allocate VMs from a collection of cloud providers. Sarastro's information service provides the following functions:

- Add, remove, and list cloud service providers (e.g. Amazon EC2 [44], Brightbox [45])
- Query cloud properties (e.g. location, CPU, storage, features, pricing),
- List available VM templates.

6.4.2 AC Management

When Sarastro creates a VM it selects and boots a VM image, bootstraps the VM, and finally uploads and executes the Sarastro agent. The VM image determines the operating system and prepared software of the VM. We prepared images with a basic Linux install on all the cloud locations we use. Cloud service providers do not yet share a single VM format on hypervisor (used to execute the VM) technology. Hence, sophisticated software distribution via a Content Delivery Network (CDN), for example, is generally not possible across cloud service providers [149]. In our implementation, we use Puppet [150] to bootstrap the

VM with the run-time dependencies of the Sarastro agent. Then the agent software and a secret key is uploaded to the VM and started. Once the agent is up, it may receive requests to execute Netscripts, create or delete tunnels (using vTun [151]), and collect monitoring information (including error reporting). In short, a Sarastro agent manages an AC.

6.4.3 UPVN collection

The UPVN collection encapsulates UPVN instances and provides a generic interface for creating and deleting UPVNs. In essence, the UPVN collection and the UPVN instances it does nothing else than maintaining references to VMs. As cloud service providers already manage VMs, the UPVN collection only needs to maintain the associations between VMs and the UPVNs they belong to.

Netapps can only control the ACs with which they are associated. As in previous work on gTBN [105], the access restriction is implemented by associating each UPVN instance with a secret key, which is required for each API call to Sarastro agents. The secret key is created by Sarastro on creation of a UPVN instance and uploaded to the instantiated VMs via SSH. Requests to the web application or agent are processed with HMAC [152] using the secret key, a salt, and timestamp. When the secret key is known to Netapps, they can communicate directly with the ACs. Multiple Netapps can control a single UPVN when they share its secret key. One application of this feature is that control over a UPVN may be distributed over multiple Netapps (see Section 6.6). Another application is that the moment of use of a UPVN can be decoupled from the moment of authorization and provisioning. This has been demonstrated for programmable networks in previous work [28].

Almost all operations on UPVNs, such as creating a tunnel, require cooperation between two or more components. Therefore, the Internet factory also provides a transaction service to ensure that such operations can be executed as an atomic unit, and are rolled back on failure.

6.4.4 Netscripts and an IPv6 Example

A Netscript is a collection of files consisting of at least one executable script. When a Netscript is scheduled for execution, Sarastro collects the ACs on which the script is about to execute, and uploads all content associated with the Netscript to the ACs and requests the agent to execute the Netscript. As Netscripts have no a priori knowledge about UPVNs, Sarastro implements the capability to substitute embedded code (using eRuby [143]) in the files before they are uploaded. This way, a Netapp can pass local and global parameters to Netscripts, such as unique identifiers for each AC (used in the example presented next) and topological information. This allows the development of programming methods and libraries in which the programming of individual ACs can be abstracted from the developer.

An example of a Netscript that operates on the whole UPVN is one that deploys and configures an IPv6 network service. Adding and removing network elements and links have a

direct impact on the configuration of addresses and routing. Rather than setting up static routes for the UPVN, which have to be revisited every time the UPVN changes, a Netapp can execute an IPv6 Netscript to setup and configure dynamic routing. We developed a Netscript for an IPv6 network service. IPv6 is well suited for UPVNs that can progressively change their topology.

IPv6 provides mechanisms for automatically configuring network interfaces with link local address. In IPv4, for example, routing protocols do not work without setting up valid network addresses on the interfaces. This forces a developer to explicitly manage all the addresses in the on-demand network, complicating the network service configuration. In IPv6, the Netscript only has to setup network unique addresses for each network element, which can be done by using the unique AC identifier provided by Sarastro. Hence, developers also do not have to manage addresses, which greatly simplifies the Netscript.

The steps of the IPv6 Netscript to setup the network then becomes (on each AC):

1. Install and verify required software (we used Quagga [153]),
2. Enable forwarding (adjust operating system configuration),
3. Generate routable address on loopback interface
   (using unique AC identifier from Sarastro),
4. Generate routing configuration (add interfaces and areas to the configuration file),
5. (Re-)start router daemon,
6. Start interface monitor. The monitor checks if an interface is added or removed and updates the routing process (when it cannot do the job itself, which was the case in our implementation).

In our experiments, simultaneously starting up the IPv6 network service on all the nodes lead to problems in route convergence. As a workaround, we introduced a random start-up time delay. Although the experiments showed that understanding routing protocols in UPVNs is crucial to maintain connectivity, this topic is beyond the scope of this thesis. The evaluation of routing protocols and their behavior in UPVNs requires further investigation and is a research topic on itself.

6.5 Evaluation

The development and prototyping of Sarastro and Netapps was exclusively done using a 2011 Apple MacBook Air laptop and Amazon EC2 and Brightbox cloud service providers. Sarastro was developed and deployed on the laptop. In total more than 6400 hours of

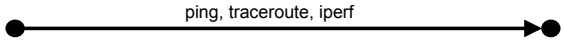
[3] https://github.com/rstrijkers/sarastro

VMs were used to implement the Internet factories and to conduct experiments. Sarastro's source code will also be made publically available[3]. We evaluate Sarastro by discussing our experience with three typical patterns to create a UPVN: an IPv6 UPVN created using the Web interface and Netscripts, a control program that creates UPVNs from a description given as input, and a dynamic UPVN in which the adaptation to link failures is described by a Netapp.

6.5.1 A Worldwide On-Demand Network

We validated Sarastro's ability to create and manage non-trivial UPVNs for a limited period of time without specific network engineering expertise. First, the UPVN was created from the Web interface by simply adding new nodes and connecting them using the Web interface. Then, the IPv6 Netscript (Section 6.4.4) was executed to create an IPv6 overlay. Finally, the UPVN was killed after verifying its operation with standard networking tools and gathering basic statistics.

In the time the experiments were executed (2012), Amazon EC2 allowed a maximum of twenty VMs at each cloud location. Therefore, the resulting UPVN of 163 nodes over 18 cloud locations in the world (see Figure 47) was a practical upper bound. The UPVN was created such that traffic over a connection from one end to the other would start in Ireland and terminate in England to form a line topology of 162 hops. Figure 46 shows the resulting UPVN in the Sarastro Web interface.



Region	eu-west	us-east	sa-east	us-west1	us-west2	northeast	southeast	gb	Total
Location	Dublin	Sao Paolo	North Virginia	North California	Oregon	Tokyo	Singapore	North England	8
Zones	3	2	3	2	2	2	2	2	18
VMs	20	20	20	20	20	20	20	23	163

Figure 47. Number of VMs created at each cloud location that together form a network topology of 162 hops around the world. The size of the UPVN was limited by the standard Amazon limitation of 20 VMs per location.

One of the issues in verifying the operational UPVN was how to test its reachability. With arbitrary topologies, all paths must be traced to find possible errors. If the network is arranged in a line topology, simply pinging the ends of the network shows its successful deployment. As it is easy to develop such a Netscript, we started with the creation of a line topology over all the possible data centers available to Sarastro. We found that to create such a line topology the Netscript had to adjust the standard hop limit in the Linux kernel to allow more than 64 hops in the UPVN.

The successful operation of the UPVN was verified with *traceroute, ping*, and *iperf* (see Figure 48). In a *traceroute* over all the virtual links, we measured 917 hops over the Inter-

net to connect the VMs. The iperf measurement show the resulting bandwidth of a TCP benchmark with long latency, large number of hops, and 7 percent packet loss, which all contribute to the result. The many hops (917) in the path also increase the chances of congested links in the UPVN, low performance transport networks, and buffer bloat. This shows that developers should be aware of the diverse quality of the virtual links in UPVNs. To a certain degree, developers can compensate for these qualities in the Netapp's control loop. But, the performance of virtual links will remain a convolution of the performance of the underlying networks.

traceroute	avg hops per link	unique routers	total routers
	7	582	917
ping	**rtt**	**avg**	**mdev**
	1668.7ms	1708.4ms	33.9ms
iperf	**bandwidth**		
	196Kbit/s		
vm boot time	**total**	**vm**	**network**
	115.7s	56.7s	58.0s

Figure 48. Results of running basic network tools traceroute, ping, and iperf over the on-demand network, and the average VM boot time.

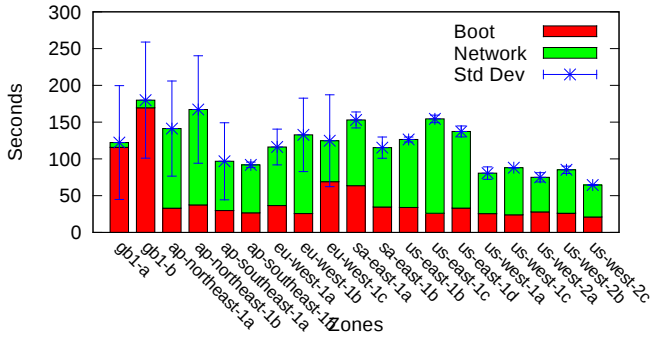


Figure 49. Startup times collected from 413 VMs including the variance. The various cloud locations show significant differences in startup time.

Our Sarastro implementation also gathered basic statistics about booting VMs. The boot time in Sarastro is a combination of two measurements: the time it takes for a cloud service to report the IP address of the VM and the time it takes until a VM accepts SSH sessions. Figure 49 shows these times for 413 VMs over the cloud locations. Remarkable is that in Amazon data centers (Figure 49, all zones except gb1-a and gb1-b) the VMs come up fast, but the time until a VM accepts an SSH session takes up 70 percent of the average Amazon VM boot time. In Brightbox data centers (Figure 49: zone gb1-a and gb1-b), however,

mapping a public IP address to private VM IP address must be setup by the end-user, so the configuration time also depends on the speed of the API calls to configure the addresses. As we have no access to the internal workings of their networks, we can only speculate about possible optimizations.

```
{
  "nodes" : {
  "n1" : ["us-west-1a"],
  "n2" : ["us-east-1b"],
  "n3" : ["eu-west-1a"]
  },
  "links" : {
    "e1" : ["n1", "n2"],
    "e2" : ["n2", "n3"]
  },
  "netscript" : ["ipv6_router"]
}
```

Figure 50. A control program was developed that creates a UVPN from a JSON description containing nodes, their location, and interconnection. The `netscript` variable indicates the desired network service, in this case IPv6.

6.5.2 Creating Netapp Templates

We developed a control program that creates a UPVN from a network description. The implementation of the control program is basically a Netapp, which can be parameterized. Instead of creating the nodes and links as part of the Netapp, developers can just pass a description of their network to Sarastro. The description is formatted in JSON [154] and defines the location of the network elements, their interconnection, and which Netscripts to execute for the desired network service (Figure 50).

The startup time of the UPVNs were measured over an average of three runs (Figure 51). Here, the created network topologies were also arranged in a line topology for automated verification of the operational UPVN (when from one end the other end is reachable by an IPv6 ping). The time to execute the control program is split in two phases: (1) creating the nodes and links of the UPVN and (2) the convergence of the routing protocol set up by the IPv6 Netscript. With the small network sizes, total start-up time and convergence time of OSPF (using default timer configurations) remained constant, which can be seen in the end-to-end IPv6 reachability results.

The number of concurrently running operations to create network elements was limited to 15 to avoid Amazon EC2 denying service due to a flood of service requests. The number was determined experimentally and impacts the speed at which UPVNs can be created and adapted by Sarastro. Clearly, cloud service providers impose such limits to avoid denial of service attacks. We regard such limits as an implementation detail as it is a trade-off between pricing of resources for access and control of VMs and pricing of resources of the VMs themselves.

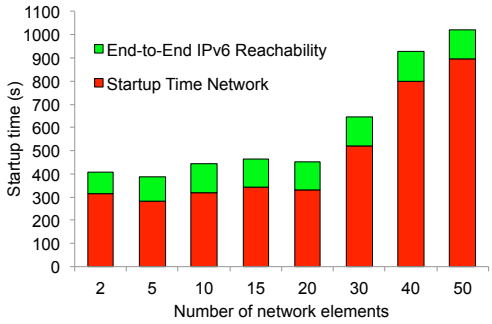


Figure 51. We developed a control program that creates UPVNs increasing in size. The average startup times over three runs per network size are shown.

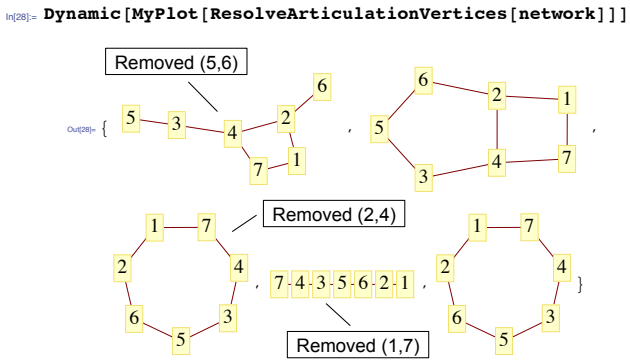


Figure 52. We visualized the UPVN topology in Mathematica while it automatically resolved single points of failure. The initial UPVN topology is displayed in Figure 53 (top right).

6.5.3 Automatically Solving Single Points of Failure

We developed a Netapp that adjusts the topology of a UPVN to desired properties, e.g. to contain no single points of failure. Instead of developing the algorithms to find and solve single points of failure, the control loop was implemented in a general-purpose scientific computing environment, called *Mathematica* [75]. Mathematica provides a wealth of mathematical, computational, and visualization software, which can be used to address network specific problems. For example, the function `ArticulationVertices[G]`, returns the single points of failure in graph G. Following previous work [41], we implemented the interfaces of Sarastro in Mathematica. Using those interfaces, Mathematica is capable of controlling UPVNs managed by Sarastro. A simple algorithm was developed to solve articulation vertices by adding new edges, i.e., by finding and connecting bi-components in the network. The implementation in Mathematica, including the topology visualization, was less than 20 lines of code. While the function `ResolveArticulationVectices[]` was

running, it visualized the UPVN topology on each event from Sarastro (Figure 52).

We created a UPVN of seven nodes (small but non-trivial topology) using the previously shown control program (see UPVN topology in Figure 53). Mathematica was instructed to receive events from the created UPVN. Then, using the Sarastro Web interface, we randomly removed links while measuring the ping times between two nodes (nodes 3 and 6). The UPVN topology allows three alternative paths. So, when a link is removed and an alternative path is available, the routing protocol takes another path. From the results, it can be observed that OSPFv3 takes 40 seconds to converge to the new topology (again, using default timer configurations). Consequently, after removing and adding the third link, the two nodes are shortly unreachable.

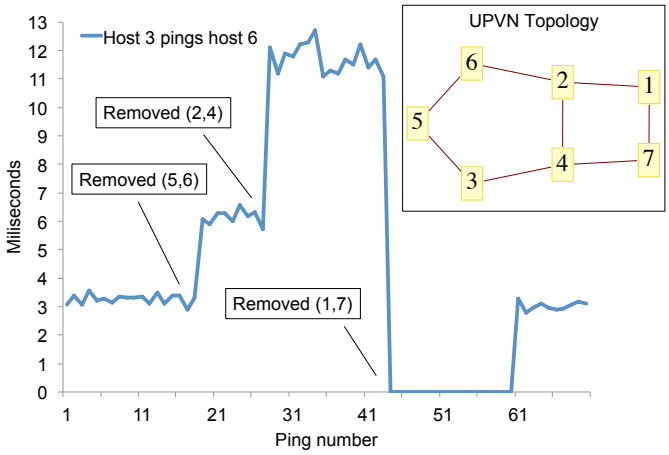


Figure 53. Ping results of a Netapp compensating the removal of links from a UPVN.

6.6 Discussion

In general, Netapps use control loops to keep UPVNs within certain operational limits. The design space of the control loops in Netapps and their multi-layer dependencies are largely unexplored. For example, designers of Netapps create algorithms that decide when and where to add ACs. When a Netapps adds or removes an AC, the network protocol will also need to adapt to the change as well. We showed how to create a Netapp in which the state of the network topology triggers an action to create new links. After a link is created, the routing protocol still has to discover and converge to the new state. If the Netapp does not take timing of the protocol into account, parts of the network can become unreachable due to inconsistent state while converging (see Figure 53). In addition, increasing the number of network elements may introduce controllability problems and require decentralization of the control loop. In any system, architectural consequences can be expected when the system becomes increasingly complex [155]. The design of control loops that adapt with

the dynamic scaling of UPVNs is a challenge. It will require both theoretical knowledge about the controllability of networks and practical experience in systems and network engineering.

In the implementation of our proof of concept, we implicitly created a platform or middleware. We assumed that infrastructure-as-a-service clouds and even future Intercloud platforms would be sufficient as a run-time platform. But our Internet factory also encapsulated state and run-time complexity of operational UPVNs that all Netapps shared (see Section 6.4.3). This suggests that there is another distinct component between the Intercloud platform and Internet factory. The question is if separating the state and run-time components of our proof of concept would define a different system, such as an operating system for Netapps.

In this research, we exploited infrastructure-as-a-service clouds for the creation and management of UPVNs. The more options a developer has to access and control computer and network resources, the better UPVNs can be made to match application-specific requirements. Further research is needed to understand at which places in the network the integration is required and most effective. For example, high quality services may only be achievable with precise application-specific control over network and computing resources with the increasing density of cells, bandwidth, mobility, and connections in telecom networks [5]. In the extreme case, cloud services might expand control into telecom antennas as well. If so, will there still be a distinction between cloud and telecom networks?

6.7 Conclusion and Future Work

Both academia and industry acknowledge that the fundamental technologies to create application-specific networks on-demand are understood (see Section 6.1 and 6.2). With the concept of the Internet Factory and our implementation of it, we showed how these fundamental technologies could be combined into a framework that structures the creation and management of applications-specific networks. Internet Factories allow developers to re-use years of experience and knowledge in network and systems engineering via software libraries. We also showed three typical patterns to create UPVNs using infrastructure-as-a-cloud services. The evaluation of three Netapps and the remarkable networks they produced underlines the design and engineering power of Internet factories. Furthermore, our use of infrastructure-as-a-service clouds demonstrates how UPVNs can use the physical Internet and how programmable network elements and networks can be part of the Internet even if network operators do not facilitate the programming of their Internet routers. Even more, these cloud services make the introduction of new network services, protocols, and architectures less costly and complex as they are defined in software and encapsulated by VMs.

We have shown how to create Netapps that adapt an overlay infrastructure towards a desired state. This can be done as a reaction to certain (networking) events but also for other reasons. Networks that continuously migrate over the Internet and cloud locations are

harder to attack, for example. Indeed, this work shows how to deal with distributed systems where one cannot abstract network details away.

We think that our use of Mathematica only reflects a glimpse of advanced systems behavior that can be created. We expect that large and complex ICT systems, such as cyber physical systems, future networks, and ICT systems to support smart cities are prime candidates for applying our results. As Internet factories show how to construct complex ICT systems, our focus will shift to the generation of advanced behavior and services of these systems. In Internet factories, the control loop of a Netapp embodies the algorithms and abstractions required to create and manage such complex ICT systems and it is here where we position our future research.

7

Conclusions and Future Work

Be prepared for reality to add a few interfaces of its own.

Eberhardt Rechtin

7.1 Conclusions

Long before the academics and industry invented Network Function Virtualization (NFV) and Software Defined Networks (SDN), the work described in this thesis already showed the practical feasibility of their underlying concepts. For example, one of the possible methods to process packets from application programs presented in UPVN is now the cornerstone of SDN technologies. The industry is hoping that programmable networking technologies will consolidate the plethora of protocols, technologies, and architectures created over the years. Without a common concept for programming the application and network domain, the best networks engineers can provide are sophisticated end-to-end connections.

To exploit the flexibility of programmable networks, the network operator needs in-depth knowledge about the application domain. From the application developer's perspective, this paradox applies as well, as choices about where to place application code, what traffic to filter, where to redirect traffic, and how to recover from failures all depend on network state and technologies. The only alternative is that both the network and application domain support a common concept for programming the application and network domain.

This thesis shows that the concept of Internet factories make the complexity of programmable networks manageable for application developers and network engineers. The concept of Internet factories envisions two types of programs. One type of program, the Netapp, is responsible for creating and deploying customized Internets on a programmable network. The other type of program, the distributed application(s) itself, uses the produced Internet and is initiated and configured by the Netapp. The result is an environment in which crosscutting concerns between networks and applications are defined, as the Internet as a whole, including its applications, is described in one program: the Netapp.

The Internet factories concept is the result of the research problem addressed in this thesis: *how to create a methodical process for programming computer networks from application programs*. The following sections reiterate the research questions and developed concepts to solve the research problem. Conclusions are drawn from their contributions.

1) *What are the common patterns for the design of applications that use programmable networks and how can these patterns be described in an architectural framework?*

The research in Chapter 2, which pre-dates the work of SDN and NFV, identifies elementary programmable networking concepts with which developers can exploit the entire design space of programmable networks and related technologies. The resulting architectural framework, called User Programmable Virtualized Networks (UPVN), describes elementary components (ACs and NCs), functions, and services to create application programmable networks.

In this thesis, the UPVN architectural framework was successfully used as a scientifically validated reference framework for creating, analyzing, and using programmable networks from application programs. The telecom industry is currently faced with the consolation of a plethora of protocols, services, and architectures using programmable network technolo-

gies, such as SDN and NFV. Also in this context, UPVN can be used as a common reference framework for creating, analyzing, and using solutions based on programmable networking.

2) *How can programmable networks support multiple application programs that control networks in application-specific ways?*

Generalized Token-Based Networking (GTBN) provides a solution for partitioning a network infrastructure into multiple distinct programmable networks. GTBN uses cryptographically encoded tokens embedded in application-generated traffic to provide applications their own programmable network environment. The use of such tokens allows dynamic binding of network services and their run-time deployment and configuration on a network infrastructure, possibly crossing multiple domains. The research presented in Chapter 3, 4 and 6 also demonstrates GTBNs capability to associate any granularity of programmable network behavior with specific applications: from application-specific paths (such as in Chapter 4) to the creation and management of complete virtualized IPv6 networks (Chapter 6). The conclusion is that sooner or later, SDN and NFV technologies will implement GTBN. Applications will have to be bound to virtual networks and using existing protocol information is simply not possible for differentiation of applications, users, and tailor-made services in the network domain.

3) *How can programmable networks be controlled from application programs, and how do controllers affect application design?*

The integration of network management in a distributed system is characterized by a control loop that collects data about network state, infers decisions, and configures nodes in a programmable network. Whether the control loop is centralized or distributed, the total amount of (distributed) state managed by a control loop affects application design. The controller is part of application programs, in later work described as Netapps, with which developers express operational boundaries of the application. The amount of detail of network state expressed in the controller reflects the network dependencies of an application program. Consequently, the scalability of network is limited by what can be managed by the application program's control loop.

We have also shown the consequence of hidden control loops in programmable networks. For example, the use case in Section 6.5.3 shows the implementation of a control loop in which new links are created when the UPVN contains single points of failure. The hidden control loop is the routing protocol that keeps topology information up-to-date. The presented application does not take into account the time it takes to converge after each intervention. The general conclusion from this example is that developers need to understand the controllability of a network, such that actions lead to a desired state. This is an essential topic for further study.

4) *How to structure the development, deployment, and management programmable network applications that make crosscutting concerns manageable?*

The implementation of any UPVN starts with an empty canvas of programmable network elements and an application program controlling those network elements. In previous work, UPVNs, even simple ones, had to be created from scratch (see Chapter 2-4). Chapter 6 introduces the Internet factories concept that combines the previous contributions to answer the research problem. The Internet factories concept describes the methodical process to program network infrastructure from application programs, embodied by *Netapps*. Implementations of Internet factories enable Netapps developers to use standardized components and common patterns, i.e. software libraries, in the development, deployment, and management of specific Internets. A developed Internet Factory proof of concept, called Sarastro, established that essentially the network is just a manifestation of the Netapp. Therefore, the contribution of Internet Factories can also be summarized as: *the Netapp is the network.*

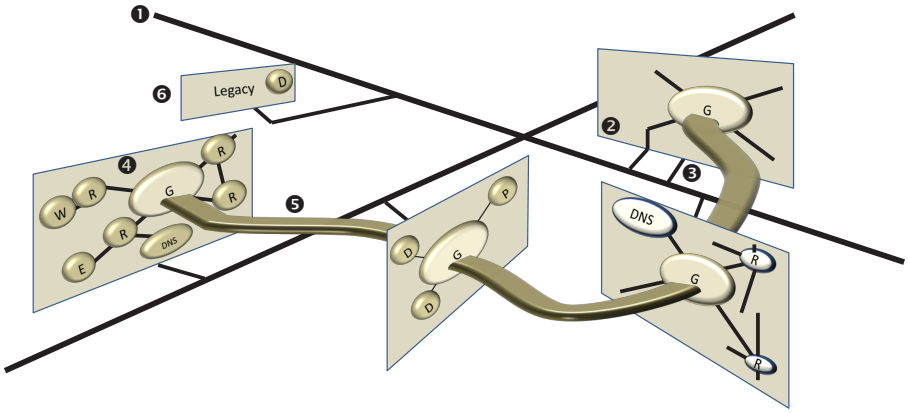


Figure 54. A complex ICT system created from several, distributed, application parts contained by virtual Internets.

7.2 Future Work

Research in Netapps and Internet factories is continuing in at least two directions. In the first research direction, the research focus shifts from programmable network technologies to the design and implementation of Netapps. How to create Netapps that can scale to the size of the Internet itself? In other words, what are the controllability and programmability limits of Netapps?

In the second research direction, further research is needed on the tooling of Internet factories. For example, tooling is required that can deal with practical issues, such as denial of service attacks, recovery from errors, and financial aspects. Tooling can also be developed to build Netapp collections (i.e., an Appstore for Netapps), which can be used to create and customize complex ICT systems. Another research question is whether Internet factories

should include optimizations for target platforms (programmable network architecture) or if Internet factories might assume a hardware abstraction layer of common functions implemented in network infrastructure.

7.3 Outlook

Figure 54 shows a scenario that becomes possible when applying the concepts presented in this thesis. This figure shows a global telecommunication infrastructure (1). The Internet (2) is just one of the complex ICT systems using (3) network infrastructures. Each complex ICT system is supported by its own specific Internet, which might consist of billions of devices (4). These complex ICT systems expose services (5) that can be used by other ICT systems. The Internet itself becomes the ecosystem to create and deploy large and complex ICT systems. The ecosystem consisting of an Internet, its infrastructure, and applications, are managed by Netapps. 54 also shows that in the future there might be many Internets, including legacy Internets (6). While some of these Internets will only support applications, other Internets will serve specific user groups, such as a children's Internet.

The described scenarios might sound extreme, but it is becoming reality. In a few years from now, the analysis of a network will reveal many Internets (see Figure 55). Some of them might have nodes that migrate from lamppost to lamppost to deliver mobile users the best possible service. Produced by Internet factories, these Internets host complex ICT systems from many collaborating organizations. They run on top of virtual machines in a highly dynamic network infrastructure continuously adapting to their operating environments.

To quote Bill Gates: 'The Internet is becoming the town square for the global village of tomorrow'. There are just town squares for many global villages and there is no single Internet that can fulfill all the needs for each global village.

Figure 55. An augmented reality app on a mobile phone shows a cross section of a telecom antenna with multiple virtual Internets [image: Marc Makkes].

Acknowledgements

Although doing a PhD is essentially a solitary effort, it requires a team of supporters to be successful. The reflection of ideas, working late hours, arranging funding, receiving feedback are impossible without support of family, friends, peers, managers, and professors. Besides the tangible result of this written work, I am proud of the time, effort, and trust that I received from my colleagues, friends, and family. On a number of occasions it took a lot of hard work of supporters as well to realize the proof of concepts of the ideas presented in this thesis.

First, I want to thank my former colleagues at the Informatics Institute. Thank you Damien Marchal, Mihai Cristea, and Leon Gommans for helping to bootstrap my research. You helped me organize vague ideas and intuitions into concrete results. One of the first results of this process, Interactive Networks, would also not be possible without the help of Laurence Muller, Willem de Bruijn, Robert Belleman, and Peter Sloot. Thank you!

At the University I enjoyed working with my colleagues, many whom I consider my friends. Thank you Adam Belloum, Reggie Cushing, Vladimir Korkhov, and Dmitry Vasyunin. When Super Computing came by, we did a lot of work in a very short time. Thanks Marc Makkes, Ralph Koning, Michiel van Tol, Rick Quax for their friendship. I thank Yuri Demchenko, Arie Taal, Leen Torenvliet, and my other UvA colleagues for the discussions and collaborations. It would have been a lot less interesting at the UvA without my students: Willem Toorop, Alain van Hoof, Michiel Appelman, Deepthi Akkoorath, Bogdan Aurelian, Ivan Mashey, Winston Haaswijk, Mohammad Shafahi, Arthur van Kleef, Marvin Rambhadjan, Alex Guirgiu, Nick Dijkshoorn, Robin Fidder, Jochem Kerkwijk, Stavros Konstantaras, George Thessalonikefs. Thank you! Each of you taught me some lessons along the way as well. As a result of such a stimulating environment, there was always a good reason to go to the University.

I thank Bernhard Plattner for having me as a visiting scientist at ETH and Wolfgang Mühlbauer and Burkhard Stiller for supporting my research while I was in Switzerland.

I also want to thank KPN. Koenraad Wuyts, Pieter Veenstra, Daphne Kreusen, Olena Butriy, and Ad Bresser, thank you for introducing me to the KPN's Long Term Research. This was my first contact with a Telco and their challenges and I learned a lot. The work at

KPN also brought me into contact with Bill Wright, Sanjay Aiyagari, and Pierre Mathys, whom I thank for broadening my views.

Without my former employer, TNO, I would not have had such a wonderful seven years. TNO was instrumental for my PhD and to gain professional experience. TNO is also a special place to do a PhD. On about any topic, there is a TNO expert. So, for me TNO was a perfect place to explore and learn. I had three research managers, Jan Jaap Aué, Richard Beekhuis, and Job Oostveen who ensured continuation of my research. Jan Jaap and Richard, thanks for letting me on board and for your support throughout my research. Thanks Job, with your support I eventually made it to the end.

TNO formed me professionally and prepared me for many things I could not learn at the University. It was only possible with coaching from my colleagues. Thank you Paul Valk, Oskar van Deventer, Jeroen Bruijning, Bjorn Hakansson, Jack Bloem, Toon Norp, and Herman Pals. Thanks to Shuang Zhang, Arjen Holtzer, Pieter Meulenhoff, Mente Konsman, Martin Nevels, Jan-Sipke van der Veen, and Lenny Zilverberg. You have been great colleagues and it was always fun to go to Groningen and Delft.

Seven years is also a long time to follow a single goal and in which much happened besides research. I met my wife on my travels with Marijn Rijken in 2007. Thanks Marijn, for the splendid idea of catching up with the Swiss ladies on the Annapurna trek! Now, seven years later I have two kids to thank as well. Maileen and Kailash, you show me how to enjoy the little things that we as adults sometimes forget. Fortunately, a scientist's mindset is not much different from that of a little child. I'm looking forward to the days we can start doing scientific experiments together! Most importantly, my wife Jasmine and my parents Shirley and Gerrit were always at my side. Thank you very much Jasmine, Shirley, and Gerrit. This thesis is dedicated to you.

Finally, I thank my supervisors Cees de Laat and Robert Meijer. Through the years, I have come to value their advice and opinions very highly. I consider myself as fortunate to have been in the SNE group under the guidance of Cees de Laat. Rob supervised my master thesis work in 2006. Since then, he has taken the role of mentor both professionally and academically. Rob, I have tried to learn as much as I can from you and I thank your commitment. Now it's time to take on new challenges and to apply and pass on the lessons with passion at Swisscom.

Overview of work

Academic Publications

A-1. Rudolf Strijkers, M. X. Makkes, C. de Laat, and R. Meijer, "Internet Factories: Creating Application-Specific Networks On-Demand," Journal of Computer Networks, Elsevier, 2014.

A-2. Rudolf Strijkers, R. Cushing, M. X. Makkes, P. Meulenhoff, A. Belloum, C. de Laat, and R. Meijer, "Towards an Operating System for Intercloud," presented at the The 3rd workshop on Network Infrastructure Services as part of IEEE Cloud Computing, Bristol, UK, 2013.

A-3. M. X. Makkes, A. Oprescu, Rudolf Strijkers, and R. Meijer, "MeTRO: Low Latency Network Paths with Routers-on-Demand," presented at the Large Scale Distributed Virtual Environments on Clouds and P2P, LSDVE 2013, Aachen, 2013, pp. 1–12.

A-4. Marc X. Makkes, Canh Ngo, Yuri Demchenko, Rudolf Strijkers, Cees de Laat, Robert Meijer, "Defining Intercloud Federation Framework for Multi-provider Cloud Services Integration", CLOUD COMPUTING 2013, The Fourth International Conference on Cloud Computing, GRIDs, and Virtualization, Valencia, Spain, 2013.

A-5. Jan Sipke Van der Veen, Mark Bastiaans, Marc De Jonge, Rudolf Strijkers: A Cloud Storage Platform in the Defense Context - Mobile Data Management with Unreliable Network Conditions. CLOSER 2012 - Proceedings of the 2nd International Conference on Cloud Computing and Services Science, pp. 462-467, Porto, Portugal, 18 - 21 April, 2012. SciTePress 2012.

A-6. Stokking, H., Rudolf Strijkers, "Sun Tzu's Art of War for telecom," 16th International Conference on Intelligence in Next Generation Networks (ICIN), 2012, pp.43-49, 8-11 Oct. 2012.

A-7. Y. Demchenko, T.C. Ngo, M.X. Makkes, Rudolf Strijkers & C. de Laat (2012). Defining Inter-Cloud Architecture for Interoperability and Integration. In The Third International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING 2012) (pp. 174-180).

A-8. Rudolf Strijkers, M. Cristea, C. de Laat, and R. Meijer, "Application framework for programmable network control," Advances in Network-Embedded Management and Applications, pp. 37–52, 2011.

A-9. Rudolf Strijkers, Reginald Cushing, Dmitry Vasyunin, Cees de Laat, Adam S.Z. Belloum, Robert Meijer, "Toward Executable Scientific Publications," in International Conference on Computational Science, ICCS 2011, Singapore: Procedia Computer Science, 2011.

A-10. Rudolf Strijkers, M. Cristea, V. Korkhov, D. Marchal, A. Belloum, C. de Laat, and R. Meijer, "Network Resource Control for Grid Workflow Management Systems," Services (SERVICES-2010), 2010 6th World Congress on, pp. 318–325, 2010.

A-11. Rudolf Strijkers, W. Toorop, A. van Hoof, P. Grosso, A. Belloum, D. Vasuining, C. de Laat, and R. Meijer, "AMOS: Using the Cloud for On-Demand Execution of e-Science Applications," presented at the ESCIENCE '10: Proceedings of the 2010 IEEE Sixth International Conference on e-Science, 2010.

A-12. Rudolf Strijkers, L. Muller, M. Cristea, R. Belleman, C. de Laat, P. Sloot, and R. Meijer, "Interactive Control over a Programmable Computer Network Using a Multi-touch Surface," presented at the ICCS 2009: Proceedings of the 9th International Conference on Computational Science, 2009.

A-13. M. Cristea, Rudolf Strijkers, D. Marchal, L. Gommans, C. de Laat, and R. Meijer, "Supporting communities in programmable grid networks: gTBN," presented at the IM'09: Proceedings of the 11th IFIP/IEEE international conference on Symposium on Integrated Network Management, 2009, pp. 406–413.

A-14. Rudolf Strijkers and R. Meijer, "Integrating networks with Mathematica," presented at the International Mathematica Symposium, Maastricht, 2008.

A-15. R. Meijer, Rudolf Strijkers, L. Gommans, and C. de Laat, "User Programmable Virtualized Networks," presented at the E-SCIENCE '06: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing, 2006.

Supervised MSc and BSc theses

T-1. M. Appelman, "Architecture of dynamic VPNs in OpenFlow," MSc. Thesis, SNE: University of Amsterdam, 2013.

T-2. D. D. Akkoorath, "Improving Quality of Experience of Internet Applications by Dynamic Placement of Application Components," MSc Thesis, FNWI:University of Amsterdam, Amsterdam, 2012.

T-3. B. Aurelian, "Operating Systems for On-demand Networks," M.Sc. Thesis, Vrije Universiteit, Amsterdam, 2012.

T-4. I. Mashey, "Discovery and Query Service for Distributed Clouds," M.Sc. Thesis, FNWI:University of Amsterdam, 2012.

T-5. W. Haaswijk, "Robust applications in the open Internet," B.Sc. Thesis, Cum laude, FNWI:University of Amsterdam, 2012.

T-6. M. Shafahi, "Bootstrapping the Internet of the Future," M.Sc. Thesis, SNE: University of Amsterdam, 2012.

T-7. A. van Kleef and M. Rambhadjan, "Self–Adaptive Routing," M.Sc. Thesis, SNE: University of Amsterdam, Amsterdam, 2010.
T-8. A. van Hoof and W. Toorop, "Grid on Demand," M.Sc. Thesis, SNE: University of Amsterdam, Amsterdam, 2010.
T-9. A. Giurgiu, "Network performance in virtual machine infrastructures," M.Sc. Thesis, SNE: University of Amsterdam, Amsterdam, 2010.
T-10. N. Dijkshoorn, "SpotSim: Test gedreven ontwikkeling van sensornetwerk applicaties," B.Sc. Thesis, FNWI: University of Amsterdam, 2009.
T-11. R. Fidder, "Applicatiespecifiek routeren in een sensornetwerk," B.Sc. Thesis, FNWI: University of Amsterdam, Amsterdam, 2009.
T-12. J. Kerkwijk, "Programmable Interplanetary Networks," B.Sc. Thesis, FNWI: University of Amsterdam, 2009.

Prototypes

P-1. Marc Makkes, Reggie Cushing, Rudolf Strijkers, Adam Belloum, Cees de Laat, Robert Meijer, Internet Factories, Super Computing 2013, Denver, Colorado. (to be submitted for e-Science 2014.)
P-2. Rudolf Strijkers, Marc Makkes, Cees de Laat, Robert Meijer, Internet Factories, Super Computing 2012, Salt Lake City, Utah. (Published in [156].)
P-3. Rudolf Strijkers, Marc Makkes, Cees de Laat, Robert Meijer, Cloud-based Interactive Networks On-demand, Super Computing 2011, Seattle, Washington. (Improved version of the 2008 demonstration.)
P-4. Rudolf Strijkers, Reggie Cushing, Dmitri Vasiuning, Adam Belloum, Cees de Laat, Robert Meijer, Intercloud Operating System, Super Computing 2010, New Orleans, Louisiana. (Published in [141].)
P-5. Rudolf Strijkers, Mihai Cristea, Robert Belleman, Cees de Laat, Robert Meijer, Generalized Token Based Networking, Super Computing 2009, Portland, Oregon. (Published in [105].)
P-6. Rudolf Strijkers, Laurence Muller, Mihai Cristea, Willem de Bruijn, Robert Belleman, Cees de Laat, Robert Meijer, Interactive Networks, Super Computing 2008, Austin, Texas. (Published in [51].)

Patents

O-1. Rudolf Strijkers, Pieter Meulenhof, Client-driven resource selection and allocation for server workload, EP13159749.4, 2013.
O-2. Rudolf Strijkers, Pieter Meulenhof, Localization and placement of servers in mobile networks, EP13159723.9, 2013.
O-3. Rudolf Strijkers, Toon Norp, PDN-GW redirection for dynamic service delivery, EP13159734.6, 2013.

Courses

C-1. M. Cristea, P. Grosso, C. de Laat, R. Meijer, and Rudolf Strijkers, "Advanced Networking - Master in Grid Computing, Computer Science track (UvA)", Academic Year 2010/2011, 2009/2010, and 2008/2009, Amsterdam.

Interviews

I-1. D. Binnendijk, "To Each His Own Internet," TNO Time, no. Winter, Delft, Dec-2013.

I-2. G. Cook, Building a National Knowledge Infrastructure. Utrecht: Cook Network Consultants, 2010, pp. 1–184.

Glossary

AAA	Authentication, Authorization, and Accounting
AC	Application Component
ACC	Application Component Collection
ACM	Application Component Manager
API	Application Programming Interface
ATM	Asynchronous Transfer Mode
BFS	Breadth First Search
CDN	Content Delivery Network
CERN	European Organization for Nuclear Research
CICS	Customer Information Control System
CORBA	Common Object Request Broker Architecture
DAS	Distributed ASCI Supercomputer
DRAC	Dynamic Resource Allocation Controller
DRb	Distributed Ruby
DTM	Distributed Transaction Monitor
FPGA	Field-Programmable Gate Array
FTP	File Transfer Protocol
GENI	Global Environment for Network Innovations
GMPLS	Generalized MPLS
GRI	Global Resource Identifier
GTBN	Generalized TBN
HMAC	Hash Message Authentication Code
ICT	Information and Communications Technology
IETF	Internet Engineering Task Force
IP	Internet Protocol
LAN	Local Area Network
LHC	Large Hadron Collider
LTE	Long Term Evolution
MPI	Message Passing Interface

MPLS	Multi Protocol Label Switching
NB	Network Broker
NC	Network Component
NCC	Network Component Collection
NCS	NC Collection Support module
NE	Network element
NFV	Network Function Virtualization
NMS	Network Management System
NOS	Network Operating System
NPU	Network Processor Unit
NUS	Network Utility Service
OGF	Open Grid Forum
OS	Operating System
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
PBT	Provider Backbone Traffic
PC	Personal Computer
QoS	Quality of Service
REST	Representational State Transfer
RFC	Request for Comments
RISC	Reduced Instruction Set Computing
RM	Resource Manager
RMI	Remote Method Invocation
RTSM	Run-Time System Manager
SDN	Software Defined Network
SOAP	Simple Object Access Protocol
TBN	Token Based Network
TBS	Token Based Switch
TCP	Transmission Control Protocol
TINA	Telecommunication Information Networking Architecture
TTS	Token Transaction Service
UCLP	User Controlled Light Paths
UDDI	Universal Description, Discovery and Integration
UDP	User Datagram Protocol
UPVN	User Programmable Virtualized Network
VLAN	Virtual LAN
VLBI	Very Long Baseline Interferometry
VLC	Video LAN Client
VM	Virtual Machine
VNE	Virtual Network Element
VNS	Virtual Network Service

References

[1] R. Dijkgraaf, C. Boonstra, T. Cuppen, W. Draijer, and W. Kuijken, Eds., "Amsterdam Institue for Advanced Metropolitan Solutions," City of Amsterdam, Amsterdam, Sep. 2013.

[2] Netherlands Organisation for Applied Scientific Research TNO and U. T. R. Laboratory, "New services enabled by the connected car," European Union, Brussels, TNO-RPT-2011- 01277, Aug. 2011.

[3] V. V. Krzhizhanovskaya, G. S. Shirshov, N. B. Melnikova, R. G. Belleman, F. I. Rusadi, B. J. Broekhuijsen, B. P. Gouldby, J. Lhomme, B. Balis, M. Bubak, A. L. Pyayt, I. I. Mokhov, A. V. Ozhigin, B. Lang, and R. Meijer, "Flood early warning system: design, implementation and computational modules," *Procedia Computer Science*, vol. 4, pp. 106–115, Jan. 2011.

[4] C. de Laat, "Kwaliteit in een Virtuele Wereld," Amsterdam, 2011.

[5] "Mobile Data Usage and Traffic | Global Mobile Data Traffic | Qualcomm," pp. 1–15, Oct. 2012.

[6] C. Schnell, Ed., "3GPP TS 32.500." [Online]. Available: http://www.3gpp.org/. [Accessed: 06-Jan-2014].

[7] V. K. Adhikari, S. Jain, Y. Chen, and Z.-L. Zhang, "How do you 'Tube': Reverse Engineering the Youtube Video Delivery Cloud," *SIGMETRICS Performance Evaluation Review*, vol. 39, no. 1, pp. 329–138, Jun. 2011.

[8] E. Nygren, R. Sitaraman, and J. Sun, "The akamai network: a platform for high-performance internet applications," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 3, pp. 2–19, 2010.

[9] A. Vakali and G. Pallis, "Content delivery networks: status and trends, " *Internet Computing, IEEE*, vol. 7, no. 6, pp. 68–74, 2003.

[10] *University of Amsterdam*. [Online]. Available: http://www.uva.nl/en/disciplines/computer-science. [Accessed: 26-Dec-2013].

[11] A. S. Tanenbaum and D. J. Wetherall, "Computer Networks, 5th edition," *Computer Networks, 5th edition*, Oct. 2010.

[12] J. D. Day, *Patterns In Network Architecture*. Prentice Hall PTR, 2008.

[13] A. T. Campbell, H. G. De Meer, M. E. Kounavis, K. Miki, J. B. Vicente, and D.

Villela, "A survey of programmable networks," *SIGCOMM Comput. Commun.* Rev., vol. 29, no. 2, p. 7, Apr. 1999.

[14] D. Hutchison, S. Denazis, L. Lefevre, and G. J. Minden, *Active and Programmable Networks*. Springer Verlag, 2009.

[15] N. Chowdhury and R. Boutaba, "A survey of network virtualization," *Computer Networks*, vol. 54, no. 5, pp. 862–876, 2010.

[16] G. Schaffrath, C. Werle, P. Papadimitriou, A. Feldmann, R. Bless, A. Greenhalgh, A. Wundsam, M. Kind, O. Maennel, and L. Mathy, "Network virtualization architecture: proposal and initial prototype," *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, pp. 63–72, 2009.

[17] R. Isaacs, "Lightweight, dynamic and programmable virtual private networks," presented at the Open Architectures and Network Programming, 2000. Proceedings. OPENARCH 2000. 2000 IEEE Third Conference on, 2000, pp. 3–12.

[18] M. Casado, T. Koponen, R. Ramanathan, and S. Shenker, "Virtualizing the network forwarding plane," presented at the PRESTO '10: Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow, 2010.

[19] A. Khan, A. Zugenmaier, D. Jurca, and W. Kellerer, "Network virtualization: a hypervisor for the internet?," *Communications Magazine, IEEE*, vol. 50, no. 1, p. 8, 2012.

[20] E. Grasa, G. Junyent, S. Figuerola, A. Lopez, and M. Savoie, "UCLPv2: a network virtualization framework built on web services," *Communications Magazine, IEEE*, vol. 46, no. 3, pp. 126–134, 2008.

[21] "Layar." [Online]. Available: https://www.layar.com/. [Accessed: 06-Jan-2014].

[22] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks, " *SIGCOMM Comput. Commun.* Rev., vol. 38, no. 2, pp. 69–74, 2008.

[23] E. Rosen, A. Viswanathan, and R. W. Callon, "Multiprotocol Label Switching Architecture," *IETF*, 2001.

[24] S. Shenker, R. Braden, and D. D. Clark, "Integrated services in the Internet architecture: an overview," *IETF RFC1633*, 1994.

[25] F. Travostino, J. Mambretti, and G. Karmous-Edwards, *Grid Networks*. Wiley, 2006.

[26] *Network Function Virtualization (NFV) Industry Specification Group (ISG).* [Online]. Available: http://portal.etsi.org/portal/server.pt/community/NFV/367. [Accessed: 03-May-2013].

[27] M. Chiosi, D. Clarke, P. Willis, A. Reid, J. Feger, M. Bugenhagen, W. Khan, M. Fargano, J. Benitez, U. Michel, H. Damker, K. Ogaki, T. Matsuzaki, M. Fukui, K. Shimano, D. Delisle, Q. Loudier, C. Kolias, V. Guardini, E. Demaria, R. Minerva, A. Manzalini, D. Lopez, F. J. R. Salguero, F. Ruhl, S. Sen, C. Cui, and H. Deng, "Network Functions Virtualization," SDN and OpenFlow World Congress,

Darmstadt, Germany, Oct. 2012.
[28] P. Bosch, A. Duminuco, F. Pianese, and T. L. Wood, "Telco clouds and Virtual Telco: Consolidation, convergence, and beyond," *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, pp. 982–988, 2011.
[29] E. M. Royer and C.-K. Toh, "A review of current routing protocols for ad hoc mobile wireless networks," *Personal Communications, IEEE*, vol. 6, no. 2, pp. 46–55, 1999.
[30] R. Newton, Arvind, and M. Welsh, "Building up to macroprogramming: an intermediate language for sensor networks," presented at the IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks, 2005, pp. 37–44.
[31] R. Sugihara and R. K. Gupta, "Programming models for sensor networks: A survey," *Transactions on Sensor Networks (TOSN*, vol. 4, no. 2, pp. 1–29, Mar. 2008.
[32] D. L. Tennenhouse and D. J. Wetherall, "Towards an active network architecture," presented at the DARPA Active NEtworks Conference and Exposition, 2002. Proceedings, 2002, pp. 2–15.
[33] S. Bhatia, M. Motiwala, W. Muhlbauer, Y. Mundada, V. Valancius, A. Bavier, N. Feamster, L. Peterson, and J. Rexford, "Trellis: A platform for building flexible, fast virtual networks on commodity hardware," *Proceedings of the 2008 ACM CoNEXT Conference*, p. 72, 2008.
[34] Y. Zhu, R. Zhang-Shen, S. Rangarajan, and J. Rexford, "Cabernet: connectivity architecture for better network services," *Proceedings of the 2008 ACM CoNEXT Conference*, p. 64, 2008.
[35] R. Sherwood, M. Chan, A. Covington, G. Gibb, M. Flajslik, N. Handigol, T.-Y. Huang, P. Kazemian, M. Kobayashi, J. Naous, S. Seetharaman, D. Underhill, T. Yabe, K.-K. Yap, Y. Yiakoumis, H. Zeng, G. Appenzeller, R. Johari, N. McKeown, and G. Parulkar, "Carving research slices out of your production networks with OpenFlow," *SIGCOMM Computer Communication Review*, vol. 40, no. 1, pp. 129–130, Jan. 2010.
[36] H. Bos, W. De Bruijn, M. Cristea, T. Nguyen, and G. Portokalidis, "FFPF: fairly fast packet filters," vol. 4, pp. 347–363, 2004.
[37] *Linux Netfilter*. [Online]. Available: http://www.netfilter.org. [Accessed: 04-Jun-2013].
[38] S. Blake, D. Black, M. Carlson, E. Davies, and Z. Wang, "An Architecture for Differentiated Services," *IETF*, 1998.
[39] S. Dobson, S. Denazis, A. Fernández, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli, "A survey of autonomic communications," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 1, no. 2, pp. 223–259, Dec. 2006.
[40] H. Zimmermann, "OSI Reference Model--The ISO Model of Architecture for Open Systems Interconnection," *Communications, IEEE Transactions on*, vol. 28, no. 4,

pp. 425–432, 1980.
[41] R. Strijkers and R. Meijer, "Integrating networks with Mathematica," presented at the International Mathematica Symposium, Maastricht, 2008.
[42] *Mathematica*. [Online]. Available: http://www.wolfram.com/. [Accessed: 09-Jan-2014].
[43] R. Strijkers, "The Network is in the Computer," University of Amsterdam, Amsterdam, 2007.
[44] *Amazon Elastic Compute Cloud (Amazon EC2)*. [Online]. Available: http://aws.amazon.com/ec2/. [Accessed: 19-Feb-2013].
[45] *Brightbox: high performance and flexible cloud servers*. [Online]. Available: http://brightbox.com/. [Accessed: 19-Feb-2013].
[46] M. Armbrust, I. Stoica, M. Zaharia, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. A. Patterson, and A. Rabkin, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, p. 50, Apr. 2010.
[47] M. Olsson, S. Sultana, S. Rommer, L. Frid, and C. Mulligan, *SAE and the Evolved Packet Core: Driving the Mobile Broadband Revolution*. Academic Press, 2009.
[48] R. Strijkers and P. Meulenhoff, "Localizing and placement of network node functions in a network ," EP13159723.9.
[49] R. Strijkers and T. Norp, "Redirecting a client device from a first gateway to a second gateway for accessing a network node function," EP13159734.6.
[50] R. Strijkers and P. Meulenhoff, "Allocating resources between network nodes for providing a network node function," EP13159749.4.
[51] R. Strijkers, L. Muller, M. Cristea, R. Belleman, C. de Laat, P. Sloot, and R. Meijer, "Interactive Control over a Programmable Computer Network Using a Multi-touch Surface," presented at the ICCS 2009: Proceedings of the 9th International Conference on Computational Science, 2009.
[52] G. Cook, "ICT and E-Science as an Innovation Platform in The Netherlands; A National Research and Innovation Network," 2009.
[53] I. Foster and C. Kesselman, *The Grid 2: Blueprint for a New Computing Infrastructure*, 2nd ed. San Francisco, CA: Morgan Kaufmann Publishers Inc., 2003.
[54] R. Strijkers, W. Toorop, A. van Hoof, P. Grosso, A. S. Z. Belloum, D. Vasuining, C. de Laat, and R. Meijer, "AMOS: Using the Cloud for On-Demand Execution of e-Science Applications," presented at the ESCIENCE '10: Proceedings of the 2010 IEEE Sixth International Conference on e-Science, 2010.
[55] R. Strijkers, R. Cushing, D. Vasyunin, C. de Laat, A. S. Z. Belloum, and R. Meijer, "Toward Executable Scientific Publications," *Procedia Computer Science*, vol. 4, pp. 707–715, Jan. 2011.
[56] J. Elischer and A. Cobbs, Eds., *FreeBSD Netgraph pluggable network stack*. [Online]. Available: http://www.freebsd.org/. [Accessed: 04-Jun-2013].
[57] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click modular router," *ACM Transactions on Computer Systems (TOCS)*, vol. 18, no. 3,

pp. 263–297, 2000.
[58] N. Dijkshoorn, "SpotSim: Test gedreven ontwikkeling van sensornetwerk applicaties," FNWI: University of Amsterdam, 2009.
[59] M. Cristea, P. Grosso, C. de Laat, R. Meijer, and R. Strijkers, "Advanced Networking - UvA Studiegids - Vak beschrijving 2010-2011." Amsterdam, pp. 1–2, 17-Feb-2014.
[60] W. De Bruijn, H. Bos, and H. Bal, "Application-Tailored I/O with Streamline," *ACM Trans. Comput. Syst.*, vol. 29, no. 2, pp. 1–33, May 2011.
[61] T. Anderson, R. Gopal, L. L. Yang, and R. Dantu, "Forwarding and Control Element Separation (ForCES) Framework," 2004.
[62] *Cisco Active Network Abstraction.* [Online]. Available: http://www.cisco.com/en/US/products/ps6776/index.html. [Accessed: 15-Jul-2013].
[63] A. S. Tanenbaum and M. van Steen, Distributed Systems: *Principles and Paradigms,* 2nd ed. Upper Saddle River, NJ: Pearson-Prentice Hall, 2006.
[64] *Forwarding and Control Element Separation (forces).* [Online]. Available: http://datatracker.ietf.org/wg/forces/charter/. [Accessed: 03-Jun-2013].
[65] E. Haleplidis, R. Haas, S. Denazis, and O. Koufopavlou, "A Web-Services Based Architecture for DynamicService Deployment," *IWAN2005, France*, pp. 1–6, 2005.
[66] E. Haleplidis, R. Haas, S. Denazis, and O. Koufopavlou, "A web service-and ForCES-based programmable router architecture," *IWAN2005, France*, 2005.
[67] G. Hjálmtysson, "Architecture Challenges in Future Network Nodes," presented at the IWAN2005, France, 2005.
[68] C. E. Perkins, *Ad hoc networking*. Addison-Wesley Professional, 2001.
[69] E. Bonabeau, M. dorigo, and G. Theraulaz, *Swarm Intelligence : From Natural to Artificial Systems*. Oxford University Press, USA, 1999.
[70] L. Gommans, C. de Laat, and R. Meijer, "Token based path authorization at interconnection points between hybrid networks and a lambda grid," presented at the Broadband Networks, 2005. BroadNets 2005. 2nd International Conference on, 2005, pp. 1378–1385.
[71] S. van Oudenaarde, Z. Hendrikse, F. Dijkstra, L. Gommans, C. de Laat, and R. Meijer, "Dynamic paths in multi-domain optical networks for grids," *Future Generation Computer Systems,* vol. 21, no. 4, pp. 539–548, Apr. 2005.
[72] D. Flanagan and Y. Matsumoto, *The Ruby programming language*. O'Reilly Media, 2008.
[73] F. Dupuy, G. Nilsson, and Y. Inoue, "The TINA consortium: toward networking telecommunications information services," *Communications Magazine, IEEE,* vol. 33, no. 11, pp. 78–83, 1995.
[74] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction To Algorithms*. MIT Press, 2001.

[75] *Mathematica*. [Online]. Available: http://www.wolfram.com/mathematica/. [Accessed: 02-May-2013].

[76] *CICS*. [Online]. Available: http://www-01.ibm.com/software/htp/cics/. [Accessed: 03-Jun-2013].

[77] F. Travostino, P. Daspit, L. Gommans, C. Jog, C. de Laat, J. Mambretti, I. monga, B. van Oudenaarde, S. Raghunath, and P. Yonghui Wang, "Seamless live migration of virtual machines over the MAN/WAN," *Future Generation Computer Systems*, vol. 22, no. 8, pp. 901–907, Oct. 2006.

[78] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden, "A survey of active network research," *Communications Magazine, IEEE*, vol. 35, no. 1, pp. 80–86, 1997.

[79] J. E. Van der Merwe, S. Rooney, I. Leslie, and S. Crosby, "The Tempest-a practical framework for network programmability," *Network, IEEE*, vol. 12, no. 3, pp. 20–28, 1998.

[80] D. S. Alexander, W. A. Arbaugh, A. D. Keromytis, and J. M. Smith, "A secure active network environment architecture: realization in SwitchWare," *Network, IEEE*, vol. 12, no. 3, pp. 37–45, 1998.

[81] *Global Environment for Network Innovations.* [Online]. Available: http://geni.net. [Accessed: 05-Jun-2013].

[82] R. Meijer, R. Strijkers, L. Gommans, and C. de Laat, "User Programmable Virtualized Networks," presented at the e-Science and Grid Computing, 2006. e-Science '06. Second IEEE International Conference on, 2006, p. 43.

[83] M. Cristea, L. Gommans, L. Xu, and H. Bos, "The token based switch: per-packet access authorisation to optical shortcuts," presented at the NETWORKING'07: Proceedings of the 6th international IFIP-TC6 conference on Ad Hoc and sensor networks, wireless networks, next generation internet, 2007.

[84] Y. Demchenko, A. Wan, M. Cristea, and C. de Laat, "Authorisation infrastructure for on-demand network resource provisioning," presented at the Grid Computing, 2008 9th IEEE/ACM International Conference on, 2008, pp. 95–103.

[85] D. Thain and M. Livny, "Parrot: Transparent User-Level Middleware for Data-Intensive Computing," *Scalable Computing: Practice and Experience*, vol. 6, no. 3, Jan. 2001.

[86] G. Goth, "Major Players Battle over Layers: Layer-2 and Layer-3 Vendors Tussle over Metro Links," *Internet Computing, IEEE*, vol. 11, no. 5, pp. 7–9, 2007.

[87] M. G. Reed, P. F. Syverson, and D. M. Goldschlag, "Anonymous connections and onion routing," *IEEE J. Select. Areas Commun.*, vol. 16, no. 4, pp. 482–494, May 1998.

[88] W. Sun, G. Xie, Y. Jin, W. Guo, W. Hu, X. Lin, M.-Y. Wu, W. Li, R. Jiang, and X. Wei, "A cross-layer optical circuit provisioning framework for data intensive IP end hosts," *Communications Magazine, IEEE*, vol. 46, no. 2, 2008.

[89] T. Lehman, J. Sobieski, and B. Jabbari, "DRAGON: a framework for service

provisioning in heterogeneous grid networks," *Communications Magazine, IEEE*, vol. 44, no. 3, pp. 84–90, 2006.

[90] C. Phillips, J. Bigham, L. He, and B. Littlefair, "Managing dynamic automated communities with MPLS-based VPNs," *BT technology journal*, vol. 24, no. 2, Apr. 2006.

[91] J. Maassen and H. Bal, "Smartsockets: solving the connectivity problems in grid computing," presented at the HPDC '07: Proceedings of the 16th international symposium on High performance distributed computing, 2007.

[92] *NuFW - An Authenticating Firewall*. [Online]. Available: http://www.nufw.org. [Accessed: 05-Jun-2013].

[93] *OpenBSD AuthPF*. [Online]. Available: http://www.openbsd.org/faq/pf. [Accessed: 05-Jun-2013].

[94] D. A. Joseph, A. Tavakoli, and I. Stoica, *A policy-aware switching layer for data centers*, vol. 38, no. 4. ACM, 2008, pp. 51–62.

[95] H. Bos, R. Isaacs, R. Mortier, and I. Leslie, "Elastic network control: An alternative to active networks," *Journal of communications and networks*, vol. 3, no. 2, pp. 153–163, 2001.

[96] D. Kindred and D. Sterne, "Dynamic VPN communities: implementation and experience," presented at the DARPA Information Survivability Conference & Exposition II, 2001. DISCEX '01. Proceedings, 2001, vol. 1, pp. 254–263.

[97] *The Distributed ASCI Supercomputer 3*. [Online]. Available: http://www.cs.vu.nl/das3. [Accessed: 05-Jun-2013].

[98] V. Korkhov, D. Vasyunin, A. Wibisono, V. Guevara-Masis, A. S. Z. Belloum, C. de Laat, P. Adriaans, and L. O. Hertzberger, "WS-VLAM: towards a scalable workflow system on the grid," presented at the WORKS '07: Proceedings of the 2nd workshop on Workflows in support of large-scale science, 2007.

[99] R. Meijer and A. R. Koelewijn, "The Development of an Early Warning System for Dike Failures," *1st International Conference and ...*, 2008.

[100] N. Kruithof, "E-Vlbi Using a Software Correlator," in *Grid Enabled Remote Instrumentation*, no. 36, Boston, MA: Springer US, 2009, pp. 537–544.

[101] *CERN*. [Online]. Available: http://public.web.cern.ch/public/. [Accessed: 04-Jun-2013].

[102] H. B. Lim, Y. M. Teo, P. Mukherjee, V. T. Lam, W. F. Wong, and S. See, "Sensor grid: integration of wireless sensor networks and the grid," presented at the Local Computer Networks, 2005. 30th Anniversary. The IEEE Conference on, 2005, pp. 91–99.

[103] I. Foster, "Globus Toolkit Version 4: Software for Service-Oriented Systems, " *Network and parallel computing*, 2005.

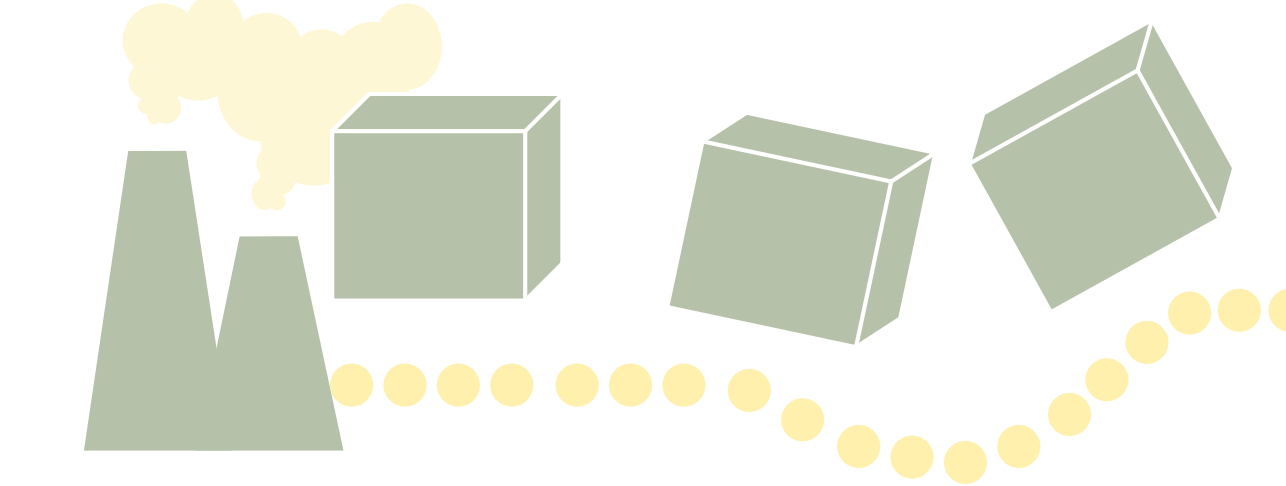
[104] *OASIS Web Services Resource Framework (WSRF)*. [Online]. Available: http://www.oasis-open.org/committees/wsrf/. [Accessed: 04-Jun-2013].

[105] M. Cristea, R. Strijkers, D. Marchal, L. Gommans, C. de Laat, and R. Meijer, "Supporting communities in programmable grid networks: gTBN," presented at the IM'09: Proceedings of the 11th IFIP/IEEE international conference on Symposium on Integrated Network Management, 2009, pp. 406–413.
[106] S. Portegies Zwart, D. Groen, T. Ishiyama, K. Nitadori, J. makino, C. de Laat, S. McMillan, K. Hiraki, S. Harfst, and P. Grosso, "Simulating the Universe on an Intercontinental Grid," *Computer*, vol. 43, no. 8, pp. 63–70, 2010.
[107] P. Wang, I. monga, S. Raghunath, F. Travostino, and T. Lavian, "Workflow Integrated Network Resource Orchestration," presented at the GlobusWorld, 2005.
[108] F. Travostino, R. Keates, T. Lavian, and I. monga, "Project DRAC: Creating an applications-aware network," *Nortel Technical ...*, 2005.
[109] H. Newman, R. Cavanaugh, J. J. Bunn, I. Legrand, S. H. Low, D. Nae, S. Ravot, C. D. Steenberg, X. Su, M. Thomas, F. van Lingen, Y. Xia, and S. Mckee, "The Ultralight project: the network as an integrated and managed resource for data-intensive science," *Computing in Science & Engineering*, vol. 7, no. 6, pp. 38–47, 2005.
[110] S. R. Thorpe, L. Battestilli, G. Karmous-Edwards, A. Hutanu, J. MacLaren, J. Mambretti, J. H. Moore, K. S. Sundar, Y. Xin, A. Takefusa, M. Hayashi, A. Hirano, S. Okamoto, T. Kudoh, T. Miyamoto, Y. Tsukishima, T. Otani, H. Nakada, H. Tanaka, A. Taniguchi, Y. Sameshima, and M. Jinno, "G-lambda and EnLIGHTened: wrapped in middleware co-allocating compute and network resources across Japan and the US," presented at the GridNets '07: Proceedings of the first international conference on Networks for grid applications, 2007.
[111] K. Yang, X. Guo, A. Galis, B. Yang, and D. Liu, "Towards efficient resource on-demand in Grid Computing," *ACM SIGOPS Operating ...*, 2003.
[112] B. Berde, A. Chiosi, and D. Verchere, "Networks meet the requirements of grid applications," *Bell Labs Tech. J.*, vol. 14, no. 1, pp. 173–184, May 2009.
[113] H. Bal, Ed., *DAS-4: Prototyping Future Computing Infrastructures*. [Online]. Available: http://www.nwo.nl/projecten.nsf/pages/2300154150_Eng. [Accessed: 04-Jun-2013].
[114] M. Hayashi, T. Kudoh, and Y. Sameshima, "G-lambda: Coordination of a Grid Scheduler and Lambda Path Service over GMPLS," presented at the Optical Communications, 2006. ECOC 2006. European Conference on, 2006, pp. 1–4.
[115] A. Bassi, M. Beck, J.-P. Gelas, L. Lefevre, T. Moore, J. Plank, and P. Vicat-Blanc Primet, "Active and logistical networking for grid computing: the e-Toile architecture," *Future Generation Computer Systems*, vol. 21, no. 1, pp. 199–208, Jan. 2005.
[116] F. Bouhafs, J. P. Gelas, L. Lefevre, and M. Maimour, "Designing and evaluating an active grid architecture," *Future Generation ...*, 2005.
[117] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: taking control of the enterprise," presented at the SIGCOMM '07: Proceedings

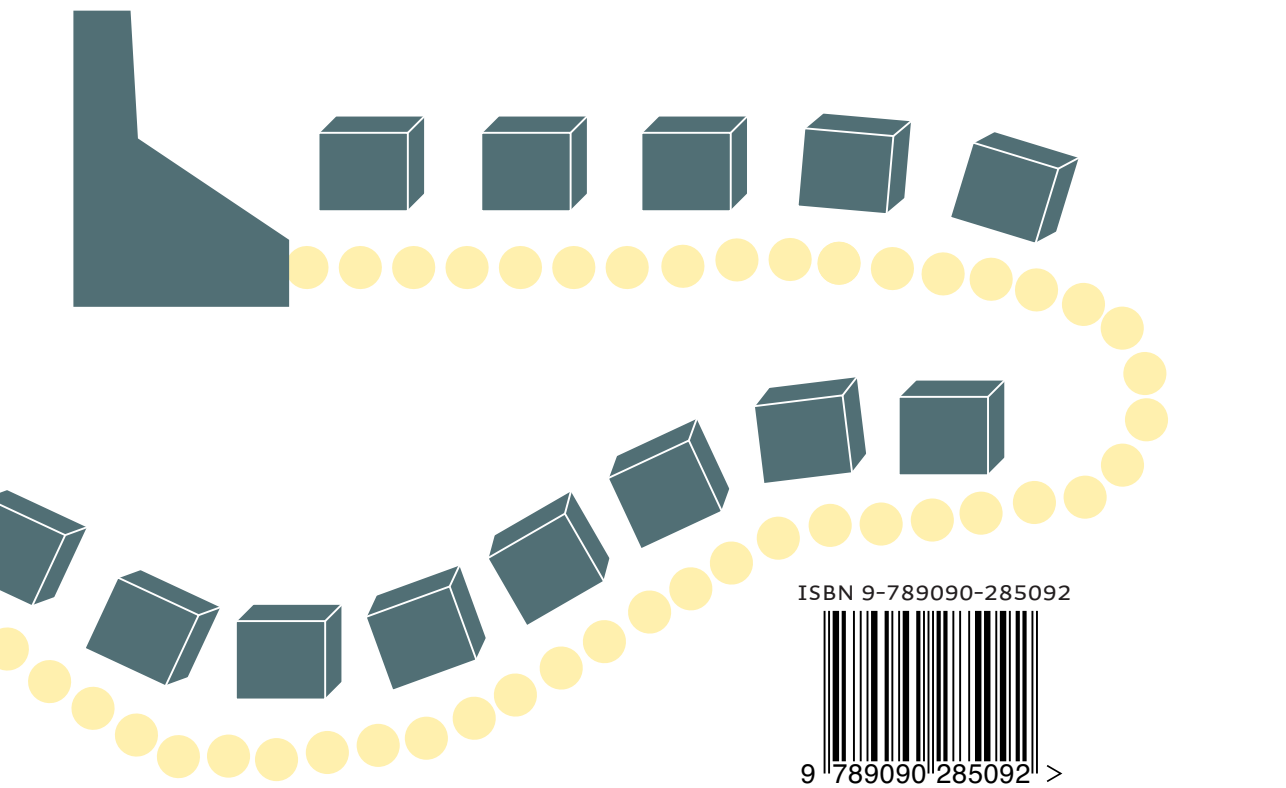
of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications, 2007.
[118] D. Culler, D. Estrin, and M. B. Srivastava, "Guest Editors' Introduction: Overview of Sensor Networks," *Computer*, vol. 37, no. 8, pp. 41–49, 2004.
[119] T. S. E. Ng and H. Yan, "Towards a framework for network control composition," presented at the the 2006 SIGCOMM workshop, New York, New York, USA, 2006, pp. 47–51.
[120] W.-M. Wang, L.-G. Dong, and B. Zhuge, "Analysis and Implementation of an Open Programmable Router Based on Forwarding and Control Element Separation," *J. Comput. Sci. Technol.*, vol. 23, no. 5, pp. 769–779, Nov. 2008.
[121] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. Gude, N. McKeown, and S. Shenker, "Rethinking enterprise network control," *IEEE/ACM Transactions on Networking (TON,* vol. 17, no. 4, pp. 1270–1283, Aug. 2009.
[122] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: towards an operating system for networks," *SIGCOMM Comput. Commun.* Rev., vol. 38, no. 3, pp. 105–110, 2008.
[123] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *Communications Surveys & Tutorials, IEEE*, vol. 7, no. 2, pp. 72–93, 2005.
[124] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," presented at the SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, 2001.
[125] *Open MPI: Open Source High Performance Computing.* [Online]. Available: http://www.open-mpi.org/. [Accessed: 04-Jun-2013].
[126] H. Xie, Y. R. Yang, A. Krishnamurthy, Y. G. Liu, and A. Silberschatz, *P4p: provider portal for applications,* vol. 38, no. 4. ACM, 2008, pp. 351–362.
[127] K. Romer and F. Mattern, "The design space of wireless sensor networks," *Wireless Communications, IEEE*, vol. 11, no. 6, pp. 54–61, 2004.
[128] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TinyDB: an acquisitional query processing system for sensor networks," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 122–173, Mar. 2005.
[129] R. Strijkers, M. Cristea, V. Korkhov, D. Marchal, A. S. Z. Belloum, C. de Laat, and R. Meijer, "Network Resource Control for Grid Workflow Management Systems," *Services (SERVICES-1), 2010 6th World Congress on*, pp. 318–325, 2010.
[130] P. Haggerty and K. Seetharaman, "The benefits of CORBA-based network management," *Commun. ACM*, vol. 41, no. 10, pp. 73–79, Oct. 1998.
[131] B. Fortz, J. Rexford, and M. Thorup, "Traffic engineering with traditional IP routing protocols," *Communications Magazine, IEEE*, vol. 40, no. 10, pp. 118–124, 2002.
[132] J. L. Hellerstein, Y. Diao, S. S. Parekh, and D. M. Tilbury, *Feedback Control of*

Computing Systems. Wiley, 2004.
[133] *VMWare*. [Online]. Available: http://www.vmware.com. [Accessed: 04-Jun-2013].
[134] *OpenVPN*. [Online]. Available: http://www.openvpn.net/. [Accessed: 04-Jun-2013].
[135] M. Cristea, W. De Bruijn, and H. Bos, "FPL-3: Towards Language Support for Distributed Packet Processing," *NETWORKING 2005 Networking ...*, vol. 3462, no. 60, pp. 743–755, 2005.
[136] *Network Mapper*. [Online]. Available: http://nmap.org. [Accessed: 04-Jun-2013].
[137] D. G. Feitelson and L. Rudolph, "Distributed hierarchical control for parallel processing," *Computer*, vol. 23, no. 5, pp. 65–77, 1990.
[138] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "Network Configuration Protocol (NETCONF)," *IETF*, 2011.
[139] R. Strijkers, M. Cristea, C. de Laat, and R. Meijer, "Application Framework for Programmable Network Control," in *Advances in Network-Embedded Management and Applications*, no. 3, Boston, MA: Springer US, 2011, pp. 37–52–52.
[140] M. Pióro and D. Medhi, *Routing, Flow, and Capacity Design in Communication and Computer Networks*. Morgan Kaufmann, 2004.
[141] R. Strijkers, R. Cushing, M. X. Makkes, P. Meulenhoff, A. S. Z. Belloum, C. de Laat, and R. Meijer, "Towards an Operating System for Intercloud," presented at the The 3rd workshop on Network Infrastructure Services as part of IEEE Cloud Computing, Bristol, UK, 2013.
[142] M. X. Makkes, A. Oprescu, R. Strijkers, and R. Meijer, "MeTRO: Low Latency Network Paths with Routers-on-Demand," presented at the Large Scale Distributed Virtual Environments on Clouds and P2P (LSDVE 2013), Aachen, 2013, pp. 1–12.
[143] D. Thomas, C. Fowler, and A. Hunt, *Programming Ruby 1.9*. Pragmatic Bookshelf, The, 2009.
[144] A. Harris and K. Haase, *Sinatra*. Oreilly & Associates Inc, 2011.
[145] T. Macedo and F. Oliveira, "Redis Cookbook," *Redis Cookbook*, Aug. 2011.
[146] W. Beary, Ed., *The Ruby Cloud Services* Library. [Online]. Available: http://fog.io/. [Accessed: 19-Feb-2013].
[147] *EventMachine*. [Online]. Available: http://eventmachine.rubyforge.org/. [Accessed: 01-May-2013].
[148] D. Recordon and D. Reed, "OpenID 2.0: a platform for user-centric identity management," *Proceedings of the second ACM workshop on Digital identity management*, pp. 11–16, 2006.
[149] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl, "Globally distributed content delivery," *Internet Computing, IEEE*, vol. 6, no. 5, pp. 50–58, 2002.
[150] J. Turnbull, "Pulling Strings with Puppet: Automated System Administration Done Right," *Pulling Strings with Puppet: Automated System Administration Done Right*,

Feb. 2008.
[151] M. Krasnyansky, "vTun Virtual Tunnel." [Online].
Available: http://vtun.sourceforge.net/. [Accessed: 09-Nov-2013].
[152] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed Hashing for Message Authentication," Feb. 1997.
[153] *Quagga Routing Software Suite*. [Online].
Available: http://www.nongnu.org/quagga/. [Accessed: 19-Feb-2013].
[154] "JSON (JavaScript Object Notation)." [Online]. Available: http://www.json.org/. [Accessed: 10-Nov-2013].
[155] M. E. Csete, "Reverse Engineering of Biological Complexity," *Science*, vol. 295, no. 5560, pp. 1664–1669, Mar. 2002.
[156] H. Rudin, Ed., "Internet factories: Creating application-specific networks on-demand," *Computer Networks*, 2014.



This thesis contributes a novel concept for introducing new network technologies in network infrastructures. The concept, called Internet factories, describes the methodical process to create and manage application-specific networks from application programs, referred to as *Netapps*. An Internet factory manufactures and deploys Netapps, each with their own set of technologies and services. To create Netapps developers use standardized components and common patterns, i.e. software libraries, capturing years of experience and knowledge in network and systems engineering. In essence, the Netapp describes the life cycle of a network from its creation, operation, to its decommissioning. Therefore, the contribution of this thesis can be summarized as: *the Netapp is the network*. It is in the design and implementation of Netapps that we find new challenges in network research for years to come.



ISBN 9-789090-285092