ANA Project
Autonomic Network Architecture

Sixth Framework Programme
Priority FP6-2004-IST-4
Situated and Autonomic Communications (SAC)

Project Number: FP6-IST-27489

Deliverable D.1.1 - State of the Art

Network Architecture Design

Authors
(alphabetical order)

Christophe Jelger and Sylvain Martin

Version: 1.0

ANA Project

Autonomic Network Architecture

Project Number	FP6-IST-27489
Project Name	ANA - Autonomic Network Architecture
Document Number	FP6-IST-27489/WP1/D.1.1
Document Title	State of the Art - Network Architecture Design
Workpackage	WP1
Editor	Christophe Jelger (UBasel)
Authors	Christophe Jelger (UBasel) Sylvain Martin (ULg)
Contributors	Stefan Schmid (NEC) Manolis Sifalakis (ULancs)
Reviewers	Guy Leduc (ULg) Rudolf Roth (FOKUS) Christian Tschudin (UBasel) Lidia Yamamoto (UBasel)
Dissemination level	Public
Contractual delivery date	31^{st} June 2006
Delivery date	30^{th} August 2006
Version	Version 1.0

Abstract:

This document compiles and analyses previous and recent research efforts in the area of network architecture design. The goal for ANA is to extract key design principles and mechanisms that can potentially become fundamental building blocks of our future autonomic network architecture.

Keywords:

Internet Architecture, Network Architecture, Design, Active Networking.

Executive Summary

The goal of the ANA project is to explore novel ways of organising and using networks beyond legacy Internet technology. The ultimate goal is to design and develop a network architecture that can demonstrate the feasibility and properties of autonomic networking. This new network architecture shall not only scale in size but also in functionality such that the core of the network is capable to evolve. This document compiles and analyses previous and recent research efforts in the area of network architecture design.

Designing a network architecture implies identifying and specifying the design principles of the system being developed. The architecture defines the atomic functions and entities that compose the network and specifies the interactions which occur between these various building blocks. In the past, various network architectures have been proposed and the Internet emerged as the "winner" architecture. Its design follows a set of principles, namely the end-to-end arguments, global connectivity, and the best-effort datagram forwarding paradigm. While these fundamental principles led to the deployment success of the Internet, a growing part of the research community now considers that they also prevent any innovative evolution of the core of the Internet.

Various modifications to the current Internet architecture or, more ambitiously, alternative network architecture designs (i.e. so-called "clean-slate" designs) have been proposed in the past. While these proposals introduce different abstractions and mechanisms, they share a number of architectural principles – what seems to indicate that the research community reaches an agreement with respect to some networking concepts. The "identifier and locator split" is a widely agreed concept and so is the principle that "independently managed addressing realms" can co-exist. In the mean time, the "active networking" community has developed techniques in order to build programmable networks, enabling flexible and potentially evolvable network architectures.

According to many indicators, "the time is right" to develop a radically new network architecture. In Europe, ANA is timely positioned to explore the design space of autonomic networking, and so are other projects of the EU-funded initiative in Situated and Autonomic Communications (SAC). In the US, the GENI initiative aims at developing a generic world-wide research infrastructure upon which next generation network architectures could be built and tested. Compared to other projects, ANA will focus on the "autonomic" behavior of future network architectures: this is where ANA will provide the majority of its innovations.

Table of contents

Chapter 1

Introduction

The main objective of the ANA Project is to develop an autonomic network architecture. As specified in the project description of work [1], we intend to address the self-* features of autonomic networking such as self-configuration, self-optimization, self-monitoring, self-management, self-repair, and self-protection.

Autonomic networking is a very recent research area. Although many publication titles already contain the "autonomic" keyword, the number of reference papers in this area is very limited. In particular, there does not yet exist an autonomic network architecture and the ANA project is doing the spadework on this wide and challenging research topic. In a first step, we want to identify the most prominant networking trends that have been widely accepted by the research community, and then extend these core concepts with autonomic capabilities. The goal is not to "re-invent the wheel", but rather to develop "an autonomic wheel" from existing best-practices and know-how in networking research.

This document compiles and analyses previous and recent research efforts in the area of network architecture design. The goal for ANA is to extract key design principles and mechanisms that can potentially become fundamental building blocks of our future autonomic network architecture.

1.1 What is a network architecture?

A network architecture is a set of design principles describing the scope, the objectives, and the abstract operation of a communication system. It acts as the master plan of the network that is to be developed as it organises the *vision* of the designers into a coherent framework. The architecture defines the atomic functions and entities that compose the network and specifies the interactions which occur between these various building blocks. Similar to the concept of *structuralism*, the core objective when designing a network architecture is to define the relationships between the fundamental elements upon which a higher structure (i.e. protocol entities, layers, the network) is built.

One can note that the "precision" at which past and current network architectures have been defined can greatly vary. Some network architectures solely define a set of high-level principles and goals while other consider details such as packet formats and implementation directives. Moreover, defining a network architecture also implies outlining what is not part of the architecture, e.g. what is left open to implementors. For example the Internet architecture specifies that IP is the addressing protocol while the OSI networking model specifies that there is a network layer but does not mandate any specific addressing scheme.

In the past, the specifications of historical network architectures such as Digital's DECNET, IBM's SNA, Xerox's XNS, and a wide collection of public switched telephony networks provided fine-grained implementation details but usually lacked high-level principles and definitions. Clearly the practical implementation aspect was favored. Oppositely, the OSI networking model [2] in the late 70ies has specified a layered set of abstract architectural design principles and guidelines with the aim of allowing interoperability between systems from different manufacturers. The detailed internal operation of each system was out of the scope of the OSI standard. Similarly, and more recently, network architectures (see Chapter 4) are merely a collection of abstract principles which often lack concrete implementation details. In ANA, we aim at finding a good balance between a high-level "blueprint" of the architecture and more detailed implementation-oriented specifications. This document is restricted to helping fulfill the first part of the objective and hence purposely focuses on design principles.

We also want to mention that ANA will take into account *emerging* network architectures such as wireless ad hoc and sensor networks as well as delay-tolerant networks. However, this state of the art document does not reference proposals in these research areas. The main reason for this omission is that these recent networking trends have produced mechanisms and protocols that, for the most part, adhere to the existing Internet design principles. Also these research fields have not yet produced alternative network architectures that are mature enough to be widely accepted by the research community. Note that the bibliographical section of this document already encompasses a wide range of networking paradigms that covers many aspects of these emerging frameworks. Our goal is that the specifications of ANA will be smart and broad enough in order to allow emerging networking paradigms to be implemented under the ANA framework.

1.2 Terminology: autonomic computing, communications, and networking

As witnessed with the advent of many research projects and initiatives that include the word 'autonomic' in their title, a growing research community is forming around this new research field. Unfortunately, it is clear that there does not yet exist a clear definition of what an autonomic system is and what properties it should exhibit. To clarify that situation, a recent paper [3] from members of the

ANA project proposes a set of definitions for 'autonomic' systems and advocates the need for a "test" that will enable the assessment of autonomic behavior in a system. In this section we borrow some of the text and arguments of [3].

The label "autonomic systems research" goes back to the *Autonomic* ***computing*** [4] initiative from IBM. In 2001, IBM decided to develop 'autonomic' computer systems inspired by the operation of the autonomic nervous system in biological systems. IBM's initiative specified four properties of autonomic systems: self-healing, self-protection, self-configuration, and self-optimisation. However, IBM's vision appeared to be limited to the development of software applications and frameworks that could auto-configure, download updates, and occasionally learn user preferences or automatically adapt to usage patterns. It seems that with IBM's initiative the premise of self-managed and self-organising network architectures have been considered but was not actively pursued.

The next step in autonomic systems research was triggered by a white paper from Smirnov [5] in the framework of the Autonomic **Communications** research initiative [6]. Extending IBM's initial concepts, Smirnov considers how self-aware independent network elements collaborate in order to fulfill the general goal of the overall system to which they belong. The major contribution with respect to IBM's initiative is that network elements are aware of the overall system goal: they are designed to continuously sense their environment in order to resolve potential conflicts by constantly adapting their operation. The focus here is on the (global) collaboration of atomic elements that take local behavioral corrective actions in order to steer the overall system towards its operational goal.

In 2005, the European Union published a call for proposals under the proactive initiative in Situated and Autonomic Communications (SAC) which led to ANA. The focus of the ANA project is on autonomic **networking**. It is a part of the more broader autonomic communications field and is concerned with a practical application of autonomic principles. The goal is to design and develop a framework that enables the flexible, dynamic, and fully autonomic formation of atomic functional blocks into network nodes, services, and entire networks. The architectural challenge is to design the generic substrate upon which autonomic principles and mechanisms can be developed and tested. The architecture must allow for, in reaction to changing networking conditions, the dynamic adaptation and reorganisation of the network elements according to high-level directives. Only the capacity of the network to be polyfunctional and fully adaptable will justify the label 'autonomic'.

1.3 Document organisation

This state of the art document starts in Chapter 2 with recapitulating the existing Internet architecture. We briefly review the fundamental design principles of this network architecture and provide a large number of reference papers related to the design of the Internet. We then introduce a few articles that highlight some of the

main limitations of the current Internet architecture, including papers from early designers of the Internet. We then mention a selection of expected features of a future Internet as proposed in the literature.

In the following Chapter 3, we present research proposals that intended to "patch" the current Internet architecture in order to overcome some of its limitations that result from inherent design principles. Note that we have attempted to limit this chapter to proposals that do not advocate a fully alternative network architecture when compared to the existing Internet.

In the fourth chapter of this document, we present alternative network architectures which are often cited in the literature. We have tried to restrict this chapter to proposals which introduce significant changes to the existing Internet such as changes to the original design principles (e.g. end-to-end connectivity or the end-to-end design principle). We believe that this chapter presents a set of networking mechanisms that significantly encompass a wide range of alternative network design proposals.

Chapter 5 provides an overview of the active networking paradigm. It reviews a selection of the prominant proposals of this research field and situates them in the new context of autonomic networking. More specifically, this chapter relates work in active networking to some of the self-* properties of autonomic networking.

Finally, Chapter 6 provides a synthesis of the document. The objective is to extract key principles and lessons from the past 30 years of experience in network architecture research. Of particular interest to ANA, we identify networking concepts for which the research community has reached consensus. We then mention the research fields that this project will have to consider when developing ANA, list related projects in autonomic networking, and finally recap techniques usually used to test and validate a network architecture.

Note that whenever possible and reasonable, papers are referenced in chronological order.

Chapter 2

The Internet

In the 1970ies and 80ies, the Internet protocol suite was one contender among many others. Thirty years later, the Internet has become the ultimate internetwork connecting millions of computers throughout the world. In many aspects of both its design and deployment, the Internet is an incomparable success. In this chapter, we discuss some of the architectural principles and design choices that have predominantly shaped the Internet. These principles are seen as the fundamental building blocks that led to the success of the Internet. One objective of this document is to better understand to which extent these central design aspects made it possible for the Internet to evolve from a contender architecture to its today's predominant status. This is discussed in Section 2.1.

The Internet architecture (and its current *implementation*) has of course a number of limitations. For many researchers including some of the early Internet's designers, the salient design principles that contributed to the success of the Internet are also preventing the evolution of the network. For example, these limitations relate to the failures (in terms of global deployment) of projects like RSVP, IP multicast, mobile IP and IPv6. Section 2.2 discusses some of the issues introduced by the Internet design. The objective here is to identify areas where our autonomic network architecture should differ from the existing Internet architecture in order not to repeat the same mistakes.

In the mean time, many research teams have proposed protocols and networking schemes in order to overcome some of the limitations introduced by the Internet architecture. In Section 3, we highlight some of the proposals that could be applied to the existing Internet without requiring a complete and disruptive revision of the architecture. While it is difficult to draw a clear boundary of acceptable architectural changes, we try to restrict this section to proposals that merely apply *patches* to the current Internet architecture.

2.1 Internet Design

The Internet protocol suite has some of its roots in the ARPANET, a network architecture developed by the Advanced Research Projects Agency (ARPA) in the late 60ies and during the 70ies. During the last 25 years, the Internet has largely evolved from the initial ARPANET design. In particular, key principles of the Internet design were not existing in the ARPANET. In this section, we present some of the design decisions that led to the current Internet architecture. Our objective is to highlight the fundamental aspects of the Internet design. In the literature, two papers ([7, 8], see below) are generally refered to when it comes to describing the design of the Internet.

2.1.1 End-to-end principle

The first paper by Saltzer et al. [7] from 1984 describes what is today commonly known as the *end-to-end principle*. In short, the end-to-end principle states that, whenever possible, networking functions should be placed at the end-points of a communication system. Key arguments in favor of the end-to-end principle are reduced redundance[1] and reduced complexity of the networking subsystem. This leads to the model of a dumb network with smart terminals, as opposed to the previous paradigm (i.e. telephone networks) of a smart network with dumb terminals: it is "the rise of the stupid network" [9]. The flagship of the end-to-end principle is TCP: reliable transmission is solely achieved by the end hosts without any help from the intermediate network nodes. This differs from the initial ARPANET design [10] which used a hop-by-hop acknowledgment scheme in addition to an end-to-end acknowledgement scheme.

The end-to-end principle has driven the Internet design and deployment in a fundamental way. For many, it is the key feature of the Internet success. However, the end-to-end principle recently raised a number of debates, such as whether it prevents innovation to take place at the networking layer. This topic is further discussed in Section 2.2. When designing ANA, we should make sure to clearly understand the pros and cons of the end-to-end design principle.

2.1.2 End-to-end connectivity and best-effort datagrams

The second key paper describing the Internet design was published by D. Clark [8] in 1988. Of high interest to network designers, the paper describes the (supposed) set of goals that have driven the design of the Internet. As stated in the paper, the most important goal of the Internet was the interconnection of independent

[1] Here the term 'redundance' relates to a situation where two or more mechanisms are used to achieve some goal that could be satisfied with less mechanisms. For example, the ARPANET used both an end-to-end and a hop-by-hop acknowledgement scheme which were redudant: the end-to-end scheme alone is sufficient to achieve reliable transport.

networks. This led to the use of packet switching for multiplexing and to the deployment of gateways interconnected by a generic network layer: the Internet Protocol (IP). IP, as the common denominator for network interconnection, implied the utilization of an addressing strategy where each Internet host is configured with a globally unique address. This design led to a property known as *end-to-end connectivity* or *end-to-end addressing*: each host of the network can send packets to all other nodes of the network without requiring intermediate network elements to perform any operation but packet forwarding. For "Internet purists", end-to-end connectivity is a fundamental property of the Internet where the network becomes a dumb packet forwarding machine [9]. However and as discussed in Section 2.2, deployment realities have since long broken the Internet transparency.

According to [8], the second most important goal was that the Internet should continue to supply communication services in the face of failures (links, gateways, networks). This feature is usually called survivability. Here rather than rephrasing the original arguments, we directly use some text taken from [8] because it incisively describes this objective:

> (...) if two entities are communicating over the Internet, and some failure causes the Internet to be temporarily disrupted and reconfigured to reconstitute the service, then the entities communicating should be able to continue without having to reestablish or reset the high level state of their conversation. (...) In other words, at the top of transport, there is only one failure, and it is total partition. The architecture was to mask completely any transient failure.

The main consequence of this design objective is that the datagram was chosen as the basic building block for data delivery. Reliability of transmission was not guaranteed at the network level but followed the now well-known "best-effort" paradigm. The main argument in favor of datagrams is that they offer the flexibity to create different types of services such as reliable transfer (TCP). Clearly, the decision to use a best-effort datagram service greatly reduced the complexity of the Internet architecture, considerably facilitating both its implemention and deployment. In particular, the only state stored in intermediate switching nodes relates to routing and forwarding tables. This means that for the Internet to be resilient to failures, only routing must be able to reconfigure dynamically in order to maintain accurate packet forwarding. From the ANA perspective, Internet routing can be seen as a self-healing activity, i.e. a first step towards autonomic behavior. As stated earlier, survivability was one of the most important design goal of the architecture: the implementation clearly fulfills this objective.

2.1.3 Address allocation

As described in the previous subsections, the Internet architecture follows a number of fundamental principles, namely the end-to-end principle, end-to-end connectivity, and best effort datagram forwarding. Among these principles, end-to-end connectivity is the most constraining one as it implies that the Internet must be based on a generic network layer and, more importantly, on a global addresing

space. The main issues when dealing with a global addressing space are, on the technical side, routing scalability, and, on the administrative side, address allocation.

In the Internet, scalability is achieved (to some extent) because routing is hierachically organized. Hierarchical routing was known long before the Internet and was already studied by Kleinrock et al. in the late 70ies [11]. The main idea is to aggregate routing information in order to reduce the size of routing tables. This implies that address allocation must be performed in such a way that address aggregation is maximised.

However during the early days of the Internet, the small number of connected networks and hosts did not encourage the Internet community to allocate addresses in a conservative way. In 1992, the Internet community decided to tackle the problems of poor routing aggregation and address exhaustion (i.e. Class B subnets). This effort included a stricter address allocation strategy ([12, 13]) and the introduction of mechanisms to perform routing aggregation ([14]). The second half of 1993 marks the start of the deployment in the Internet of Classless Inter-Domain Routing (CIDR [15]) which introduces the concept of variable length subnet masks (VLSM). CIDR replaces the previous generation of IP address syntax, namely classful networks (i.e. A, B, and C subnets). This facilitates routing by allowing blocks of addresses to be grouped together into single routing table entries.

The history of address allocation in the Internet soundly illustrates that network addressing is not just a technical issue and that designing an addressing scheme implies that the scheme should be used in an effective way. However, the persons running a network might be different from the persons that designed the network. A major challenge for the designers is hence to be able to provide precise guidance to the network operators so that they can run the network in the most efficient way. While this requirement seems obvious, the history of the Internet shows that this has always been difficult to achieve [8]. Note that this issue has recently reappeared with a debate whether the current IPv6 address allocation strategy [16] is already not conservative enough [17]. In order to avoid the many pitfalls of network addressing, the ANA consortium should be extremely careful when designing the addressing framework of our autonomic network architecture. To shed light on this critial topic, Section 3 contains many references related to address assignment. We, the ANA designers, should make sure we will handle network addressing with great care.

2.1.4 Naming, addressing, and forwarding

In its current state, the Internet architecture implements a networking model that is perfectly described by the set of abstract definitions proposed in 1978 by Shoch [18] for the three fundamental networking concepts of name, address, and route:

> The *name* of a resource indicates what we seek,
> an *address* indicates where it is, and

a *route* tells how to get there.

The main advantage of these definitions is that they are very simple to identify if one considers the current Internet networking model. In particular, Shoch's paper is being refered to in [19] in order to situate IP with respect to naming, addressing and forwarding. This networking model is deeply embedded in modern operating systems and applications which largely operate in an *Internet by default* mode of operation. One of the task of the ANA consortium is to carefully analyse to which extent the predominance of Shoch's networking model is beneficial (or unfavorable) to the Internet and how it suits modern networking paradigms such as mobility, security, or sensor-based networks. For example, addresses are not always required either because the fundamental activity of a network is to resolve a name into a route [20], or because addresses are not desired [21, 22, 23]. Clearly, the Internet architecture is not flexible with respect to this issue because its addressing scheme is one of its fundamental building block. A core issue in the ANA design is hence whether ANA should impose any addressing scheme or if this should be left open in order to allow diversity.

Naming in the Internet is today dominated by the Domain Name System (DNS) which was specified in the early 80ies by P. Mockapetris [24, 25]. Prior to the DNS, names were manually managed and stored at a central location in the HOSTS.TXT file but the rapid growth of the Internet quickly highlighted the limitations of this approach. The DNS implements a distributed database in which names are stored in a hierarchically organized tree structure. Control of the database is delegated in a hierarchical fashion where management boundaries usually correspond to leaves and branching points of the tree. The DNS rapidly gained popularity and in 1987, the HOSTS.TXT file contained approximately 5,500 host names while the DNS contained about 20,000 host names [26]. Today the DNS system stores hundreds million names which are the primary abstraction via which human users interface with the Internet. However in order to operate properly, the DNS requires a very high load of human maintenance and it is at the opposite of being a self-configurable scheme. Adding autonomicity in the future naming system(s) of ANA is one of the many challenging tasks of this project.

2.2 Internet Design Debated

While the deployment of the Internet is clearly an immense success, its architecture and design principles continues to trigger debates among the networking community. Most of these critics can probably be linked to the frustration caused by the impossibility to modify the core operation of the Internet. We here refer to the failures (in terms of large-scale deployment) of projects like RSVP, IP multicast, mobile IP and IPv6, and the inability of the Internet to evolve from its original design (i.e. a dumb forwarding machine). This section summarizes some of the debates related to the Internet architecture and discusses some of the expected features of a next-generation Internet. Our objective is to identify areas where our

future autonomic network architecture could potentially depart from the current Internet.

2.2.1 Internet under stress

The Internet (in a very broad sense) has clearly evolved since its initial design and many assumptions that held 20 years ago are no longer valid. Blumenthal and Clark [27] propose a very detailed description of the many (external) changes that are putting the core Internet design principles under continuous stress. As stated in [27], many stakeholders believe that new networking requirements might be best met through the addition of new mechanisms in the core of the network. This approach is in clear contradiction with the end-to-end principle and the historical argument that the Internet should remain a dumb forwarding machine where "packets go in, and they come out, and that is all that happens in the network [28]". In [27], the authors examine many technical and political scenarios that call for new mechanisms to be deployed "in" the network. Although they acknowledge the tension generated by new networking requirements (technical, political, or purely business driven) on the Internet architecture, the authors strongly defend the end-to-end arguments. One of their main argument is that the end-to-end principle (and the general nature of the Internet) preserves innovation and flexibility in end systems, i.e. two fundamental objectives of the initial architecture (the term variability is used in [8]).

Variability was clearly achieved but solely at the application layer. This is where remarkable breakthroughs have been achieved and where diversity was obtained: DNS, email, web, VoIP, and peer-to-peer applications are the highlights to mention here. By design, the networking layer of the Internet has not aimed at functional scaling. The *waist* of the architecture, i.e. IP, the end-to-end principle, and end-to-end connectivity, was not meant to be negotiable. However, modern networking requirements and constraints (e.g. economic tussles and security, all-optical transmission, constrained sensor nodes, intermittently connected devices) are over-stretching the core concepts of the Internet architecture. In addition, the limited success (in terms of deployment) of numerous projects like RSVP, IP multicast, and IPv6, has revealed the inability of the core Internet architecture to evolve. This seriously questions the ability of the Internet to become the sole global communication network that will eventually replace existing infrastructures such as telephone, cable, and TV networks [29].

A recurrent topic in the literature is whether the core principles of the Internet should be relaxed in order to allow unconstrained innovations to appear. As a matter of fact, this is already happening. The most significant example is the widespread use of network address translation (NAT) and the resulting loss of both end-to-end connectivity and network transparency [30] (i.e. the failure of a NAT box typically results in the loss of all passing connections). Another example is the increased deployment of content caches which replicate popular data "close" to the users in order to improve the delivery performance. Another violation of

the end-to-end principle can be seen in multi-hop wireless networks where many protocols rely on a hop-by-hop acknowledgement of packets in order to cope with transmission links that exhibit a high error rate. Clark, a key designer of the Internet, admits in [28] that "the end-to-end arguments are still valid and powerful, but need a more complex articulation in today's world". We believe that such lucid statement help demystify the core design principles of the Internet which are often perceived as the untouchable axioms of the architecture. Clearly, if autonomic networking is to become a reality, one should be ready to reconsider the fundamental principles of the Internet design. Autonomicity will not appear without an added complexity cost which may well trigger the "dusk of the stupid network".

2.2.2 Properties of the next-generation Internet

While the networking community seems to converge on the fact that the Internet needs considerable changes in order to cope with emerging networking paradigms, there exist few papers that propose a *global vision* of what the future Internet might be. Most of the literature indeed focuses on a particular aspect of the Internet but not on the global architecture. We tried to restrict this subsection to papers that propose such a global vision because the ANA consortium is facing a similar challenge when designing an autonomic network architecture. Note that papers dealing with one aspect of the Internet are discussed in Section 3. Moreover, papers that propose a fully alternative architecture are discussed in Chapter 4 (we indeed note that proposing an alternative architecture is different from expressing some high-level expectations for a future Internet).

2.2.2.1 The knowledge plane

It is interesting to see that in the early days of the Internet the lack of proper management tools was already becoming a growing issue. Back in 1988, Clark [8] noted that "the most important change in the Internet architecture over the next few years will probably be the development of a new generation of tools for management of resources in the context of multiple administrations". Fifteen years later, Clark published a paper [31] in which he proposes to develop a network architecture that

> (...) can assemble itself given high level instructions, reassemble itself as requirements change, automatically discover when something goes wrong, and automatically fix a detected problem or explain why it cannot do so.

In this paper, Clark explicitly states that the fundamental design of the Internet, i.e. a dumb network with smart terminals, is the main cause of most of the frustration experienced by end users and operators. The main reason is that the network carries data without knowing what that data is and without knowing its own high-level purpose. In short, Clark's argument is that "if the network had a

high-level view of its design goals and the constraints on acceptable configurations, then it could make many low-level decisions on its own". This high-level framework is called by Clark the *knowledge plane*. The final goal of the architecture is "a network that can configure itself, that can explain itself, that can repair itself, and does not confound the user with mysteries".

According to Clark, end users are frustrated when something fails because typically the network is not capable of providing useful information about the reasons of the failure. In the mean time, operators trying to fix a problem are overwhelmed with management overhead that usually requires manual configuration, diagnosis and design. Although the paper does not propose a technical description of the knowledge plane, it sketches its expected features and capabilities. With striking similarities, the knowledge plane exhibits most of the expected properties of autonomic networking: self-awareness, self-configuration, self-protection, and self-repair. However, while the goal of the ANA project is to demonstrate autonomic networking with a real implementation and testbed, the knowledge plane has remained a visionary exercise that so far failed to materialize into a prototype.

2.2.2.2 Addressing reality

In [32], Clark introduces two networking concepts that, he believes, should guide the evolution of the next-generation Internet. These are design for change and controlled transparency.

Now generally admitted, *design for change* is a key characteristic for which a network architecture must be designed. The Internet was designed with end-to-end evolvability in mind as its non-specialized nature was expected to support any unforeseen future application. The main consequence is that Internet evolution has solely happened at the edges and the core of the architecture has failed to evolve. In ANA, we believe that the network itself should be able to evolve although this might have an impact on performance and efficiency. In contrast to common (mis)beliefs, designing a network core capable of evolution does not necessarily violate the end-to-end design principle [33]: the objective is to permit evolution, not only of the applications running on top of the network, but of the core architecture itself.

Revisiting a key principle of the Internet, namely end-to-end connectivity, Clark introduces *controlled transparency* as a key feature of the next-generation Internet. The main observation here is that the current reachability model of the Internet (i.e. "on by default") generates many security concerns such as intrusion and denial-of-service attacks. Clark proposes that the degree of trust between two communicating parties should modulate the degree of transparency at the network level. To be efficient, controlled transparency would require to be implemented "in" the network as a fundamental design principle. Note this does not necessarily violate the end-to-end arguments because controlled transparency (at the network layer) cannot be implemented solely at the edges. In order to promote flexibility and generality, Clark proposes to implement "in"-network functionalities via a

shared soft state support subsystem within the architecture.

2.2.3 How can the Internet evolve?

In Section 2.2.1 we discussed some of the reasons that are believed to restrain or even prevent the evolution of the Internet. These issues have been recently discussed in more details in two papers that also attempt to propose solutions to re-ignite evolution at the network layer. Note that the goal of the ANA project is to not to design solutions and protocols aimed at "evolving the Internet". ANA is a *clean-slate* design and it hence should not be subdued by some backwards Internet compatibility constraints. However if we, the ANA designers, want to develop a flexible and evolvable network architecture, we have to be fully aware of the current discussions related to Internet evolution.

Peterson et al. observe in [34] that current networking research is seriously limited because with today's Internet it is impossible to deploy, test, and analyse new network architectures in a sufficiently large and realistic manner. To overcome this situation, they propose that a *virtual testbed* should be built. The physical topology of such a testbed is a set of nodes and links but in contrast to existing networks, this physical substrate can host multiple isolated *slices*, each running a different network service, application, or architecture. This virtual testbed design was demonstrated via the well-known PlanetLab project. An intrepid statement in [34] is that, in a constantly evolving networking world, the virtual testbed could become the model of the Internet where new architectures always compete against the old ones and where "the strongest survive".

In a recent paper, Ratnasamy et al. [35] propose to use *anycast* in order to introduce new network protocols (e.g. a new version of IP) in the Internet. In short, anycast can allow hosts to tunnel the new protocols' packets to the nearest ISP capable of handling them. The paper discusses how such a system could be deployed in the current Internet and it illustrates this proposal with an example (i.e. the deployment of source-specific multicast). Actually the goal of this short summary is not to describe this solution and interested readers are invited to refer to [35] for more details. We believe that the main interest of the paper is actually not its technical proposal but rather the discussion on ISPs not having any incentive to see the Internet evolve. The main reason is that ISPs are much more sensitive to marketing issues than technical arguments. If they do not see any financial benefit in deploying a new protocol or architecture, it is very unlikely that they will do so even if the new mechanisms introduce important technical innovations. While this statement may seem obvious, it is a good reminder that the commercial deployment of network technologies is highly business-driven. A widespread assumption is that self-managed and self-healing autonomic systems are very likely to be adopted by ISPs because they could greatly reduce the human cost of managing networks. A critical goal of the ANA project is thus to ensure it can initiate sufficient leading research efforts to fulfill this objective in the next 5 to 10 years.

2.2.4 About network resilience

Another broad issue related to the current Internet architecture relates to resilience. The term resilience is used to describe the ability of the network to provide and maintain an acceptable level of service in the face of various challenges to normal operation, such as unusual traffic conditions, mobility of nodes and subnetworks, attacks against network components, or unavailability of network components due to mis-configuration, episodic connectivity, hardware or software failures, or large-scale natural disasters.

The original design of the Internet (ARPANET), had accounted for a certain degree and aspects of resilience, in particular by moving most state to the edges and in routing. However the way today's Internet has evolved lacks fundamental support for resilience both at the infrastructure level as well as at the service level. As the Internet evolves to a pervasive and always-on infrastructure used for critical services, and its size continues to increase to scales that are inherently difficult to manage manually, the need for mechanisms to provide resilience and survivability at all dimensions is becoming more and more evident. In the rest of this section, we briefly describe some aspects and current research proposals related to resilience.

Dependability [36] defines system properties which lead to a resilient network. Therefore threats, i.e., faults, errors and failures, dependability attributes, and the means for dependability are classified. Survivability as one of these properties is the capability of a system to fulfill its mission in a timely manner, even in the presence of attacks or failures, including large scale natural disasters. Another area is disruption tolerance networking. That is the ability for end-to-end applications to operate even when network connectivity is not strong (weak, episodic, or asymmetric) and the network is unable to provide stable end-to-end paths [37].

Closely related to resilience is fault tolerance [38], the ability of a system or component to continue normal operation despite the presence of hardware or software faults. Fault tolerant systems are generally engineered only to tolerate isolated random natural failures. Thus, fault tolerance is necessary but not sufficient for survivability (and therefore resilience). We do believe that the ANA design can learn and benefit from past work in fault tolerance, particularly by extending work in design methodology and metrics.

Many of the efforts to introduce resilience in the current internet architecture have made evident that the rigidness and opacity of the layered architecture is one of the fundamentally restricting factors in addressing the problem of resilience. Therefore a significant amount of research currently focuses on mechanisms that will relax this opaqueness by allowing cross-layer information sharing albeit without compromising the modularity of the architecture [39, 40, 41]. An area of particular interest aims in tackling the issue of feature interaction [42] and its impact on the stability of a working system that could defeat the purpose of enabling cross layer information sharing by sabotage the endeavoured benefits.

2.3 Summary

In this chapter we have reviewed the design principles of the Internet, namely the end-to-end arguments, global connectivity (and transparency), and the best-effort forwarding paradigm. We have also recalled the current naming and addressing strategies of the Internet. While these principles led to the success of the Internet, there is broad consensus in the research community that the Internet architecture is under stress as it fails to evolve at the network layer and hence hardly copes with emerging networking paradigms.

The need for more evolvability (e.g. design for change) of the network layer is a recurrent topic in the literature, as is the need for better network management frameworks (in a broad sense). This is a clear signal that ANA should address these issues, not only in a classical way but by adding autonomic capabilities to the core network. Whether the design principles of the current Internet should be preserved is controversial and is discussed in the following two chapters.

Chapter 3

Internet Patches

With the ever growing popularity of the Internet, the need for more evolution of the Internet architecture has become more obvious. This is reflected in the literature with the publication of uncountable proposals attempting to modify or optimise some aspects of the Internet. While it is of course impossible to review all these proposals, we have tried in this chapter to capture a spectrum of contributions which cover a significant range of the published material. We have tried to restrict this chapter to proposals that do not propose an alternative Internet architecture but which rather focus on one particular aspect of the current Internet. Note that while some proposals might require a significant effort to be deployed, they could still be classfied as a somehow upgraded version of the Internet. The boundary with proposals which advocate a new architecture (see Chapter 4) is of course imprecise and we do not claim that this classification is absolute nor definitive.

3.1 On addresses, identifiers, and their semantics

Since the early 1990ies, one of the probably most discussed topic among the networking community relates to the duality of IP addresses, which are used to both *identify* and *locate* a host in the Internet. In practice, this duality introduces many issues with modern networking paradigms such as mobility, multihoming, and the very large deployment of NAT, wherefore a future Internet architecture should clearly separate identification and location [32].

Addressing and naming (the process of assigning identifiers) are fundamental networking concepts that have been widely discussed and studied since the early ages of computer networks. Of course reviewing all the existing papers that discuss these topics is not feasible and would probably generate more questions than answers. Pragmatically, the goal of this section is to present a small subset of recent proposals on naming and addressing which are (somehow) related to autonomic networking. Note that this section only examines schemes that have been

considered to be close enough to the existing Internet in the sense that they could be implemented in a next-generation Internet and do not propose a completely different architecture (such proposals are discussed in Chapter 4).

3.1.1 4+4

The 4+4 architecture [43] is an attempt to extend the existing IPv4 address space while restoring end-to-end address transparency and allowing a simple and incremental deployment. 4+4 is a *NAT-extended architecture*, i.e. an architecture which assumes the existence of multiple private address realms and one public address realm (whose topology needs not be contiguous). In 4+4, each node has two 32-bit IPv4 addresses (a public and a private one) that form its 4+4 address. The public address identifies the address realm, and the private address identifies the node inside the realm. In practice, the public address of a node located in a private realm is the (global) address of a NAT router that connects the private realm to the public Internet. Nodes in the public Internet use their existing address as the public part and `0.0.0.0` as the private part. 4+4 packets are encapsulated IP packets and can hence be forwarded by routers that do not understand the 4+4 semantics. Routing inside a realm is unchanged (with respect to IPv4 routing) as 4+4 specific network operations are carried out by 4+4 routers located at the edges of the realms. The DNS is used to store and retrieve 4+4 addresses in conceptually the same manner as with IPv4. The DNS system needs not be updated since a 4+4 address can be stored as two separate A records (e.g. `_l1.foo.bar.edu` and `_l2.foo.bar.edu`).

The main advantage of 4+4 over for example IPv6 (whose main objectives are also to extend the addressing space and restore end-to-end connectivity) is that 4+4 deployment is less complex and can naturally coexist with traditional IPv4-only nodes and networks. Moreover, the existing IPv4 Internet can be used without modifications to connect 4+4 realms. The 4+4 deployment strategy is actually very close to the current deployment of NAT domains, except that 4+4 naturally restores end-to-end connectivity. The main drawbacks of 4+4 is that the processing of ICMP messages become more complex and the deployment of 4+4 requires that each node is updated with a 4+4-ready operating system (although this can be done incrementally). Moreover, applications also need to be updated in order to become fully compatible with 4+4. However, a transparent protocol translation mechanism similar to [44] allows to use non-updated applications.

3.1.2 Unmanaged internet protocol (UIP)

The unmanaged Internet protocol (UIP) was proposed by Bryan Ford [45] as a solution to the ever increasing management complexity of Internet addressing and routing. UIP is a fully self-organising network-layer protocol that implements scalable identity-based routing. It uses a flat namespace of cryptographic node identifiers and a routing framework closely related to distributed hash tables (DHT)

algorithms. UIP typically sits on top of IP and below the transport layer, but it can also operate directly over link-layer protocols such as Ethernet. With UIP, upper-layer protocols and applications manipulate identifiers that have no topological meaning (with respect to the underlying physical topology). A core assumption of UIP is that unique, uniformly distributed (wrt. the namespace), and self-certifying identifiers can be created in a decentralized manner with a cryptographic hash function. UIP also introduces the notion of virtual links, i.e. links between nodes that can only communicate by forwarding packets through one or more intermediate UIP nodes. In contrast a physical link at the UIP layer can be a lower-layer link or a (possibly multihop) IP path.

UIP routing is very closely coupled with neighborhood discovery: each node maintains a neighbor table divided into l buckets (where l is the length (in bits) of UIP identifiers) and ensures that it always has at least one neighbor in each bucket. To decide in which bucket a neighbor n_2 should be placed, a node n_1 determines the proximity $prox(n_1, n_2)$, i.e. the length (in bits) of the *longest common prefix* it shares with its neighbor's identifier: the neighbor is then placed in bucket $prox(n_1, n_2)$. To route packets to a target node n_t, a node n picks any neighbor n_n in bucket $prox(n, n_t)$: because of the way neighbor tables are organised, $prox(n_n, n_t)$ is at least equal to $prox(n, n_t) + 1$ and the neighbor n_n can determine a neighbor that is closest to n_t (wrt. the identifier namespace). This process continues until n_t is found. Intermediate nodes discovered on the path are added by n in its neighbor table. In practice, UIP uses either source routing or recursive tunneling to forward packets on virtual links, and it also implements a path optimization procedure: UIP paths are indeed longer than topological shortest paths because the identifier space is not topologically organized.

The main advantage of UIP is that it is fully self-organised and hence fits perfectly in the framework of autonomic networking. By self-organised, we mean here that there is no topologically organised addressing plan (i.e. a costly management overhead of the Internet). Actually, UIP does not use addresses but solely relies on identifiers. The routing scheme is interesting and introduces inspiring techniques such as the neighborhood organisation in buckets. However, scalability was only studied up to 10,000 nodes and the typical path stretch of UIP is 2 (with respect to the topological shortest path). Also the paper assumes that nodes can generate unique identifiers without the help of any centralized system: considering the current size of the Internet, this is probably an over-optimistic assumption.

3.1.3 The host identity protocol (HIP)

The main objective of the host identity protocol architecture (HIP) [46] is to decouple the transport layer (TCP and UDP) from the IP layer so that it no longer uses IP addresses to identify on-going communications. In addition, HIP introduces both authentication and anonymity services on top of the existing IP routing infrastructure. To achieve these goals, HIP introduces so-called host identifiers (HI) which are used to identify Internet hosts. A host can have multiple identities

(either public or anonymous), it may self-assert its own identity via public key cryptography, or it may ask a third-party entity to validate its identity. In practice, a HI is the public part of a public/private key pair and it can hence be used for both authentication and encryption. Public host identifiers should be stored in the DNS in a new RR type in order to allow domain names to be resolved into HIs. Because of the update latency of the DNS, node mobility is handled via rendezvous servers which can also be stored in the DNS system.

With HIP, higher layers and applications manipulate so-called host identity tags (HIT) when an IPv6 connection is initiated and local scope identifiers (LSI) for IPv4. A HIT is a 128-bit cryptographic hash of a HI that is assummed to be unique with a very high probability. HITs are used to identify the sender and recipient in HIP packets, and they are used by higher layers and applications to identify (IPv6) correspondants. A LSI is a 32-bit representation of a HI which only has a local scope: it is used by higher layers and applications to localy identify (IPv4) correspondants. The HIP "base exchange" [47] is a two-party cryptographic protocol used to establish a security association (encryption and authentication) between two hosts. It uses the well known Diffie-Hellman key exchange procedure. Despite its extra complexity and overhead, HIP provides crucial features that could significantly reduce various threats such as intrusion, denial-of-service, and man-in-the-middle attacks. The mechanisms used in HIP could clearly inspire the development of secure autonomic protocols where self-managed entities autonomously establish trusted communications without human intervention.

3.1.4 A layered naming architecture

Like many others, Balakrishnan et al. [48] claim that IP addresses should only be used as locators and that a dedicated namespace should be used to unambiguously identify hosts. In addition, they argue that an extra layer describing network services (in a very broad sense) should be used in order to decouple services from hosts, a proposal already sketched in the literature more than 20 years ago ([49]). Actually this paper does not propose any original contribution (in the traditional technical sense). As stated by its authors, "it is a pastiche of borrowed elements and the contribution is both the distillation of some basic principles and their synthesis into a coherent architecture". Indeed a clear interest of this paper is that it provides quite a large bibliographical section which encompasses most of the topics related to adding namespaces to the current Internet. The only technical contribution is the introduction of a generalized delegation principle, a dynamic framework that would allow resolution requests to be redirected to trusted third-parties. The goal here is for example to create "services front-doors" that could redirect requests to dedicated servers that registered particular services. However, while an example illustrates the high-level operation of such a delegation scheme, the paper does not present any implementation details. We insist that the main interest of the paper is its bibliographical section.

3.2 End-to-end connectivity

A key design principle of the Internet architecture is its end-to-end connectivity: any host is globally reachable from anywhere in the network. The advantage of such a design is that the network architecture does not rely on any sort of connection setup phase that creates forwarding states in the network "on-demand": complexity and overhead are hence greatly reduced. However, the downside of this openness is that every host is vulnerable to both intrusion and denial-of-service (DoS) attacks. In practice, network address translation (NAT) and firewalls already provide mechanisms that can prevent a host from receiving unwanted traffic. However, this basic filtering typically takes place when a packet reaches an edge router of the destination network and malicious packets still consume bandwidth and resources in intermediate links and hops.

While the literature is full of proposals that address security issues that aim at reducing the amount of unwanted traffic, a very simple approach to network security is to implement a *default-off* communication model. That is, network hosts are by default not reachable unless explicitly allowed. This conservative approach can prevent hosts from receiving unwanted traffic, i.e. a straightforward solution to reducing both intrusion and DoS attacks. The cost of this increased security is twofold. First, it adds complexity and overhead to the network architecture and second, it restricts the flexibility of future application deployments. Schemes which implement this default-off behavior have been proposed (in chronological order) at the addressing layer [50] and at the routing layer [51]. Note that we do not consider reactive schemes that restrict connectivity once a (D)DoS attack has been detected (such as [52]) because they strongly rely on efficient attack detection schemes.

3.2.1 A DoS-resistant Internet

Handley et al. [50] propose to restrict the current global connectivity by separating the IP address space into a set of client addresses and a set of server addresses. The main objective is to allow clients to initiate connections towards servers, but to prevent clients or servers to initiate connections towards clients. As stated in the paper, network address translation already provides a comparable protection by hiding the internal addresses of NAT domains. The idea in [50] is to generalize this "off" principle to the whole Internet. An extra feature is that client addresses are created "on the fly" during connection setup. In effect, a client address has no global significance and can (to some extent) only be used by the target server to send back packets to the client. While this feature sounds appealing, the practical details presented in the paper are somehow ambiguous and one can argue whether this mechanism is just a re-phrased implementation of source routing. This framework also introduces a serious issue for peer-to-peer applications because it prohibits that (client) peers can talk to other (client) peers. It proposes to resolve this problem by adding a "broker service" to the architecture where clients can advertise themselves

as being globally reachable.

The most interesting features of this proposal are twofold. First the paper explicitly suggests that a generalized global addressing scheme is not necessarily required for the Internet to properly operate. Second, it briefly introduces the concept of a "broker service" where nodes can advertise themselves as being globally reachable. While not fully explored in this paper, this later feature inspired the following proposal where hosts explicitly advertise their presence via a distributed reachability protocol and framework.

3.2.2 Off by default!

Ballani et al. [51] propose to implement the default-off behavior at the routing layer. The main goal of their paper is to explore the feasibility of adding reachability constraints (i.e. states) in the network. Their basic proposal is that routers should not forward packets unless explicitly directed to do so by the destination host via a *reachability protocol* that is intentionaly decoupled from whatever routing protocol is used in the network. To become reachable, hosts send reachability advertisements that propagate through the network. For scalability reasons, reachability states are aggregated by using Bloom filters: filtering is fine-grained when close to a destination and less precise as the hop distance to this destination increases, leading to imperfect filtering. Unwanted traffic can be allowed by routers located far from the destination until it is dropped when it encounters a sufficiently unaggregated filter. Filtering is hence a tradeoff between scalability and accuracy. To further reduce the overhead of this scheme and similar to [50], "off" hosts can receive packets in response to traffic they initiate without having to be "on" (hence greatly reducing the number of reachability advertisements).

Conceptually, this proposal turns the network into a global and dynamic firewall. The paper analyses (via simulations) the scalability and performance of the proposal and shows that the scheme could effectively filter most unwanted traffic in the network core for a reasonable overhead in terms of resource overhead (message processing and memory requirements). However, security issues are not addressed and the authors did not build a prototype to demonstrate the feasibility of their scheme in real networks. Nevertheless, the "default-off" paradigm would perfectly fit to autonomic networking, in the sense that it would provide a robust protection to the self-managed communication entities of the network (hosts, services, routers). The "autonomic" paradigm would require that network entities explicitly and autonomously establish a communication context (forwarding states but also authentication and encryption from e.g. [46]) before data can be sent through the network. This could greatly reduce the amount of unwanted traffic (either attacks or DoS traffic) received by hosts.

3.3 Indirection layer

One of the requirements on the Internet today that was not part of the original design is known as *indirection*. By design, the Internet provides a generalized forwarding service that (blindly) delivers packets from the source end-point to the destination end-point. The simplicity of this communication model greatly contributed to the success of the Internet by keeping complexity to a minimum ([9]). However, many modern applications would benefit from a more flexible communication abstraction that would inherently provide a layer of indirection that decouples the sending hosts from the receiving hosts, allowing packets to be redirected to one or many third-party entities. For example, indirection enables such (emerging or growing) applications as IP mobility, proxies (e.g. web caching), and multicast. To deploy an indirection service, most of the existing research proposals use application-layer solutions (e.g. peer-to-peer and overlay mechanisms) because this is where the design flexibility of the Internet is the highest. However, while most of these proposals achieve the desired functionality, these solutions are usually application-specific and do not provide a generalized indirection service. Since this document focuses on network design, we deliberately ignore task-specific application-layer solutions for indirection. We hence present two alternative designs: the first introduces a generalised indirection service on top of IP at the application layer, while the second proposes to implement an indirection service in a virtualized link-layer below IP.

3.3.1 Internet indirection infrastructure (i3)

The Internet indirection infrastructure (i3) [53] is implemented as an overlay network on top of IP. It can be used for a variety of communication services such as multicast, anycast, mobility, and service composition. In essence, i3 decouples the act of sending from the act of receiving: sources send packets to a logical identifier and receivers express interest in packets sent to that identifier. Identifiers can either be public or private, and a rendezvous-based mechanism is used to "interconnect" senders and receivers: receivers insert *triggers* of the form *(id,addr)* in the i3 infrastructure to indicate that packets sent to the identifier *id* should be forwarded to the IP address *addr*. To achieve multicasting, multiple receivers can add a trigger for a given identifier to which multiple sources can send. To achieve mobility, a node simply needs to update its triggers to indicate its new location (i.e. its new IP address). More complex communication services (e.g. "in the fly" data re-encoding) can also be achieved by using identifier stacks (see [53] for details).

In practice, the i3 system is an overlay network which consists of a set of servers that store triggers and forward packets. The overlay uses the self-organising Chord lookup protocol to map triggers to servers. However, while the overlay provides some robustness to server failures, it introduces a non-negligeable delay before the triggers of a faulty server can be reinserted in other servers. Moreover, i3

introduces many security issues which, to be removed, increase the complexity of the architecture. Routing is also sub-optimal because packets are delivered to their destination(s) via an intermediate i3 server. Nevertheless, the introduction of a logical layer on top of the physical addressing is interesting and in-line with the wide-spread opinion that a next-generation Internet should separate identifiers and locators. The contribution of i3 in that respect is that an identifier should not be restricted to identifying a single host, but that it could also be used to identify a group of nodes and/or intermediate processing agents.

3.3.2 Indirection in lower-layers

Selnet (Selector Network) [54] is a virtualized link layer implemented at layer 2.5 (below IP) that provides explicit indirection hooks to the network layer (and to higher layers if required). Selnet is based on the *network pointers* approach detailed in [55]. It provides a completely flat topology where network layer addresses are mapped into opaque labels that identify network processing functions (e.g. packet forwarding, encryption, compression, etc). Selnet labels are called *selectors*: a selector is a 64-bit flat value that identifies a function inside a given network node. Selectors only have local significance and are typically dynamically assigned, and they are usually closely coupled with resolution schemes (e.g. link-layer or name resolution). For example in the Internet, when a node resolves a DNS name into an IP address, this address immediately identifies a forwarding path to the destination host: forwarding and routing are fully independant from name resolution. In contrast with Selnet, forwarding states are typically instantiated by the resolution/lookup scheme (e.g. DNS, DHT lookup, broadcast): the path is resolved and setup at the same time. This is similar to setting up a virtual circuit, although long-haul links could use well-known and long-lived selectors to reduce the overhead of setting up such virtual circuits (see [54] for details).

In essence, Selnet provides a low-level forwarding scheme supporting indirection that is decoupled from naming and addressing. Although this is not mentioned in [54], Selnet could be incrementally deployed at the condition that Selnet-enabled nodes bypass Selnet functions when communicating with traditional IP-based hosts. Providing indirection below the network layer is a very flexible design that allows higher-layers and applications (current or future) to setup a customized packet forwarding and procesing path on-demand. The main drawbacks of this approach relate to scalability and security. Scalability relates to state maintenance and state aggregation, two "classical" problems of systems that use virtual circuits. Node lookup in a flat topology was already demonstrated (to some extent) in [45] and DHT-based schemes and is hence not the main issue. Security relates to the ability to remotely create and delete states via a resolution/lookup scheme, i.e. an open door to malicious threats such as denial-of-service attacks and silent eavedropping. Nonetheless, this approach is appealing because it somehow radicalizes the end-to-end principle in the sense that addressing is taken out of the low-level forwarding subsystem. This allows any future network architecture (and even multiple competing ones) to operate on top of a generalized low-level infrastructure.

In the area of autonomic networking, this could allow self-managed network entities to create efficient customized communication structures on-demand. It could also permit to dynamically insert processing functions into a path in an autonomic way, for example to apply security patches when an attack is detected.

3.4 Summary

As can be seen with the various proposals reviewed in this chapter, the research community has proposed a very wide range of modifications to the current Internet architecture. A recurrent topic - since more than 10 years - is the strict separation of identifiers and locators, and more recently the need for cryptographically secured identities. This implies the introduction of highly distributed lookup schemes for the handling of a namespace that a priori is flat. This is clearly a research area where ANA should propose self-configurable and self-managed secured naming schemes.

In addition, there is a clear trend to redesign the addressing and forwarding mechanisms. If end-to-end connectivity is achieved at higher layers, the network layer does no longer need to provide global reachability. That is, independently managed addressing realms (i.e. today's NATed domains) can explicitly become part of the network architecture. Moreover, global reachability becomes a modal property and could be made optional in order to avoid undesired traffic to waste network resources. Since at the scale of the Internet this can not realistically be configured by human users, autonomic mechanisms will be required to turn on connectivity on an on-demand basis. This could be achieved with the help of indirection frameworks which allow the design of more flexible communication paradigms than just best-effort packet forwarding. The challenge for ANA is to ensure that such future designs will be able to run autonomously with minimum human intervention.

Chapter 4

Alternative network architectures

This chapter presents network architectures that propose an alternative design with respect to the current Internet. As stated earlier, we have tried to restrict this chapter to proposals which introduce significant changes to the existing Internet such as changes to the original design principles (e.g. end-to-end connectivity or the end-to-end design principle). This classification is of course subjective and we do not claim that it is infallible. Nevertheless we believe there were sufficient arguments to include each of these papers in this chapter.

4.1 PIP

The PIP (Paul's IP) [56] protocol is one of the first proposals that advocated the separation of identifiers and locators. PIP identifiers (IDs) are 64 bit long and are organized as a flat namespace. PIP addresses are of variable length and are hierarchically organized. To increase routing aggregation, PIP specifies that addresses are provider-rooted which means that the top-level numbers of the hierarchical addressing scheme are assigned to large network providers (e.g. by an address assignment authority like IANA). These top-level providers then become the assignment authorities at levels below the top-level and so forth. At each level, a unique number is assigned to each network entity under the authority. Hence a PIP address is the concatenation of all the numbers assigned at each level of the hierarchy. In a sense, a PIP address can be seen as a source route where each level indicates where to forward a packet.

The DNS is used to resolve names into PIP IDs and addresses. Note that a host may be multihomed so a DNS query can return multiple PIP addresses for a given name but only one PIP ID. To forward packets, PIP routers only use one level of a PIP address to identify the next-hop domain or router to which the packet should be forwarded. A field in the PIP header indicates which part of the PIP address is to be used: this field is incremented by each intermediate PIP router. Also a PIP address only needs to contain sufficient information for a packet to be routed:

for example for a local communication, a PIP address does not need to contain any numbers about the highest parts of the hierarchy. Routing at the different hierarchy levels is achieved via traditional link-state or distance-vector protocols where each entity advertises the shortest possible PIP address it has been assigned.

While PIP was never deployed, many of the mechanisms described in [56] are (to some extent) used today. For example, PIP's provider-rooted hierarchical addressing scheme is exactly how IPv6 addresses are assigned in today's Internet: IANA assigns parts of the IPv6 address space to RIRs (Regional Internet Registries such as ARIN and RIPE) and each RIR assigns sub-parts of its address space(s) to ISPs which in turn assign sub-parts to sites and end clients. This strict address allocation scheme allows the current IPv6 routing tables in the Internet DFZ (Default-Free Zone) to contain an average of 800 prefixes, compared to about 200,000 to 250,000 prefixes for IPv4. PIP also proposed to use a host address autoconfiguration scheme that looks pretty much like the stateless address autoconfiguration of IPv6 where hosts listen to router advertisements in order to learn the prefix part of their address. Finally, PIP's recursive packet forwarding scheme requires exact matches, i.e. unlike IP there is nothing like a longest-prefix match. This is somehow similar to label-switched forwarding (e.g. MPLS) except that with PIP all labels are contained in the PIP header: the "switch" is achieved by increasing the header field pointing to the current active forwarding label.

4.2 NIMROD

NIMROD (Nimrod: It Might Run One Day) [57] was proposed as a new routing architecture for the Internet and was designed to provide service-specific routing in the presence of multiple constraints. The key feature of NIMROD is that it exchanges routing information at multiple abstraction levels called *maps*. A map is a set of (either physical or logical) nodes and arcs (links) and it can be of arbitrary size and granularity (precision), ranging from a single network host to an ISP network. Network hosts (or endpoints in NIMROD's terminology) generate routes based on maps which are dynamically fetched during communication establishments. Note that maps typically contain qualitative information (bandwidth, delay, error rate, cost) in order to allow endpoints to select routes that fulfill certain quality-of-service constraints. In practice, an endpoint computes a source route towards a destination. To reduce the overhead introduced by source routing, an endpoint can setup a virtual path (called a *flow* in NIMROD) inside the network that uses locally significant labels to perform packet forwarding. In essence, this sets a label-switched forwarding path *à la* MPLS. Hence NIMROD allows end nodes to choose their own routes (through map retrieval and path setup) with maximum flexibility because with such a scheme there is no requirement for everyone to run the same routing algorithms. Note that NIMROD also separates endpoint identifiers from locators.

A key feature of NIMROD is that endpoints are in charge of computing the

routes upon which their packets should be sent. This avoids that all routers keep a complete view of the network (with respect to forwarding) since routes are computed at the edges only when required. However, it is not clear how such a scheme would perform in a real situation. Note that this scheme is somehow similar to UIP (see section 3.1.2) where routes are found on-demand. This design departs from the existing Internet that maintains up-to-date routes to all possible destinations. With NIMROD, only active routes are maintained. Another interesting point is that NIMROD has dared to propose an architecture where virtual circuits (called flows) can be dynamically setup (note that in NIMROD a flow represents a route between two endpoints). This implies that per-flow states are stored in the network, i.e. a violation of one of the most important initial design principles of the Internet. While such a per-flow design adds complexity to maintain and secure the virtual circuits, it is in line with recent proposals to turn the default network connectivity to "off" (as already discussed in section 3.2). Here we note that solving the issues related to virtual circuits maintenance is a very interesting research area for autonomic networking. This could lead to the design of a low-level self-healing forwarding subsystem on top of which many network architectures could run.

Unfortunately, the specifications of the NIMROD architecture only describe the high-level operation of NIMROD and useful details are not covered. For example, it is not clear how an endpoint can fetch distant maps: this requires some basic hints about the location of the destination but this "bootstrap routing procedure" is not precisely defined. It is also not clear how virtual circuits are (scalably) maintained.

4.3 TRIAD

TRIAD (Translating Relaying Internet Architecture integrating Active Directories) [58] is a network architecture that, although not usually classified as such, resembles a NAT-extended architecture. Its primary goal is to define an explicit *content layer* that provides scalable content routing and caching, and that integrates naming, routing and transport connection setup. TRIAD routing is based on names (e.g. a TRIAD name can be a DNS name or a URL) but forwarding is based on so-called *Internet relay tokens (IRTs)* where an IRT is a variable length field that specifies a path from the source to the destination. IRTs are derived during the name lookup process and in practice an IRT is simply a source route. TRIAD allows peers to communicate across private addressing realms because the initial lookup/routing phase is performed with names that are expected to be globally reachable.

Although TRIAD is frequently cited in the literature, we believe that it provides little contribution and has a major limitation. That is, TRIAD relies on a name-based routing protocol which is very unlikely to scale. Aggregation of DNS domains and names is very difficult because domains are not topologically organized. For example, in January 2006 [59] there were about 70 million hostnames

ending with `.com` and about 1.9 million level-2 domains (e.g. example.com). This would mean that in the best case (all hosts and subdomains of level-2 `.com` domains are topologically close), TRIAD routers would have to maintain 1.9 million routing entries and this is just for `.com`. Clearly TRIAD routing would not scale. It is however interesting to note that TRIAD, as many other proposals, advocates the use of (DNS) names as host identifiers: there seems to be strong convergence in the networking community that addresses should solely be used to indicate location (and not identity and location as in the current Internet).

4.4 IPNL

The IP Next Layer (IPNL) [60] proposal is a NAT-extended Internet architecture that has many similarities with TRIAD but that relies on different mechanisms. IPNL relies on the existing global Internet: specific routers called *frontdoors* are used to connect IPNL sites to the existing global Internet (IPNL packets are tunneled in IP packets to travel between IPNL sites). An IPNL site may contain multiple *address realms* identified by a non-necessarily unique realm number (a frontdoor has other means to distinguish realms inside its site). An IPNL address is a triplet <frontdoor address, realm number, host address> where

- the host address only needs to be unique inside the realm of the host,
- the frontdoor address only needs to be unique in the global Internet,
- the realm number only needs to be unique with respect to a given frontdoor.

IPNL uses fully qualified domain names (FQDNs) to identify hosts. Actually during the initial packet exchanges, peers use FQDNs in packet headers to address each other because IPNL addresses are not known prior to setting up a communication. When a host A wants to communicate with another host B, it performs a lookup of B's FQDN via the (unchanged) DNS. This request resolves into the address of a frontdoor of B's site and the IPNL packet can travel up to one of the destination's frontdoor (a site may have multiple frontdoors). Routing inside a site is based on both FQDNs and IPNL addresses so a frontdoor can route packets to hosts using FQDNs. Note that when host A sends its first packet towards B, it only knows its realm address. The frontdoor of A's site is responsible for adding the source frontdoor address and the source realm number. Hence when the packet reaches host B, the IPNL source address of A is known. The same procedure happens in the reverse direction so that B's IPNL address is built when B sends its first packet towards A. Once known, IPNL addresses are used for routing. Note that FQDN-based routing is only possible at the condition that a given DNS zone is associated with exactly one realm. Hence realms typically coincide with (administratively set) DNS boundaries. Also while IPNL routing initially uses FQDNs, it switches to using IPNL addresses because forwarding using a fix-length address is more efficient.

IPNL represents a significant departure from the current routing and addressing paradigm. First, FQDNs are used as identifiers and (ephemerally) as locators before IPNL addresses are built. IPNL routers hence have to maintain FQDN-based routes (aggregatable at the "dot" boundaries): routers between realms maintain zone entries (e.g. example.net) while the routers of a given realm maintain host entries (e.g. host1.example.net). In addition, IPNL routers also have to maintain routes based on IPNL addresses. The main advantage of the IPNL architecture is that it explicitly supports private realms in a transparent manner: an IPNL address contains the full topological information of the associated host (i.e. site, realm, host address). However, IPNL routers have to maintain two databases (FQDNs and IPNL addresses) and routing becomes closely coupled to the DNS. Nevertheless, IPNL introduces original mechanisms such as the dynamic generation of addresses. In the context of autonomic communications, this technique could be used to allow network nodes to communicate without any prior manual address configuration. Self-managed network entities could dynamically negotiate addresses that could be valid during the lifetime of a communication and that would become obsolete once the communication ends.

4.5 The NewArch project

The NewArch project is a DARPA funded initiative on future-generation Internet architectures. The basic goal of NewArch was to propose a *clean-slate* design for the Internet. To quote NewArch's final report [61], the question the project contemplated was the following: "if we could design a network like the Internet today, without the need to worry about migration from the existing deployed network, what approaches would we pick?" At first sight, the ANA project resembles NewArch because it also focuses on a clean-slate network design. However, ANA puts its emphasis on the "autonomic" properties of the future network architecture, i.e. the ability of the network to self-organise and self-manage itself (in a broad sense). Nevertheless, it is interesting to examine the outputs of NewArch as it examined key properties of a future Internet design.

The Forwarding directive, Association, and Rendezvous Architecture (FARA [62]) is one of the result of the NewArch project. FARA is not a fully defined network architecture but rather a *meta*-architecture that focuses on the high-level model and definitions of the netwok abstractions. As stated in [62],

> The FARA model defines an abstract set of components and the modular relationships among them, while it leaves undefined many detailed technical mechanisms and design decisions. Thus, FARA defines a class of architectures from which a variety of specific architectures can be instantiated by adding additional assumptions and defining various mechanisms.

Interesting to note, FARA does not require that end systems have addresses chosen from a single global address space. Hence autonomously managed addressing realms are explicitly allowed and supported. Regarding packet delivery, FARA

insists that the low-level packet forwarding mechanisms should be decoupled from higher layers, an argument in line with the proposal to use opaque and locally significant handles for packet forwarding [54]. As many others, FARA advocates the separation of identifers and locators by cleanly decoupling application identity from network-layer forwarding mechanisms. As in [56], FARA introduces the concept of forwarding directive (FD), i.e. a header field that contains the information needed to cause eventual delivery of packets. In practice in M-FARA (a proposed implementation of FARA) an FD is merely a source route. FARA also assumes that nodes register their FDs into a single and generic directory service; a similar technique is used to handle mobility. Although FARA's objective is to define abstract high-level principles upon which a next-generation Internet should be built, the paper fails to present innovative networking concepts and borrows much of its content from existing work (as admitted in [62]). While the authors state that the novelty comes exactly from these abstract definitions and principles, it is not clear how these contribute to innovative future network designs. Nonetheless FARA adopts previously advocated principles such as the identifier/locator split and the strict decoupling of the packet forwarding mechanisms (with respect to higher layers).

The NewArch project also led to the concept of *role-based architecture* (RBA) as defined in [63]. Instead of organising network functionalities in layers (as traditionally done), a RBA organises communications using functional units called *roles*. A role is a functional description of a communication building block that performs some specific function relevant to forwarding or processing packets. Roles are not hierarchically organised and can hence be freely interconnected to fulfill networking tasks in a flexible way. For obvious interoperability reasons, roles should be formally described (to some extent) in order for a node to be able to perform precise role matching. A simpler alternative is to define well-known role IDs with well-defined inputs and outputs syntax and semantics [63]. While RBAs appear as somehow elegant frameworks to design flexible network stacks (or heaps), implementation details can be complex and one should be careful not to restrain the flexibility at the implementation level. For example, [63] provides general high-level features on a RBA but does not venture into practical implementation details. Other (quite successful) projects such as the *x-kernel* [64] and the Click modular router [65] provide implementations that are in some way close to RBA (although this term appeared later in [63]).

Another publication related to NewArch is a routing framework called NIRA (New Internet Routing Architecture) [66]. Although this deliverable focuses on network architectures and does not consider routing protocols, we believe that NIRA provides a valuable contribution to this document. In NIRA, routing is indeed closely coupled with addressing in the sense that routing is not globally performed with a routing protocol as in today's Internet. That is, only the routers inside the *core* of the Internet exchange routing information for the whole core (e.g. with BGP). In contrast, routers outside the core only perform some basic neighbor discovery (of other routers) and do not propagate routing information. NIRA indeed assumes that topological branches linking end-sites to the core follow a strict

hierarchical addressing scheme (similar to a tree or star) that allows packets to follow a simple (but non-optimal) "valley-free" forwarding path: *up* from the source's end-site to the core and then *down* to the destination's end-site. If addresses are strictly hierarchically assigned, next-hop forwarding is reduced to a simple prefix "best matching" algorithm: a packet is forwarded either to the neighbor router that shares the longest prefix with the destination address or *up* towards the core if no match is found. In the context of autonomic networking, such combined addressing and routing schemes are very efficient because they reduce the complexity of network self-organisation to addressing and make routing straightforward. That is, self-organising nodes have to resolve one problem and not two. The downside of these approaches is to find scalable address assignment algorithms that achieve a small routing stretch (valley-free routing does not always use shortest paths).

4.6 Plutarch

Rather than being a network architecture (in the traditional sense), Plutarch [67] is a framework for the development and explicit co-existence of heterogeneous next generation networks that does mandate a one-size-fits-all approach. In a sense, Plutarch is a continuation of the "yellow book" spirit [68]. This design is in contrast to the current Internet paradigm where the original goal was to interconnect heterogeneous networks via the deployment of a global and homogeneous network layer (IP). Plutarch introduces the notion of *contexts*, each being a logical set of hosts, routers, switches, network links etc. Within a context, addresses, packet formats, transport protocols and naming services are assumed to be homogeneous. This is comparable to the (previously defined) concept of a *realm* but with maximised heterogeneity in mind (e.g. addressing and naming in two contexts can be totally different). Communications across contexts are achieved via so-called *interstitial functions*, i.e. gateway processes which map the constituents (addresses, names, etc) of one context into another context's formats. Specifically, Plutarch explicitly disapproves the use of global names and addresses.

As stated in [67], the central issues raised by Plutarch are communicating and resolving names and addresses across network boundaries (contexts). The paper itself does not provide any technical details on how this could be achieved: it merely provides possible scenarios and high-level examples of how communications could be setup across contexts. The main contribution of Plutarch is the argument that contexts should be made *explicit*, i.e. their specificities along with the set of interstitial functions should not be hidden by the inter-networking protocol suite. That is, the heterogeneity of the network is fully revealed in order to accurately reflect the reality of the physical deployment. The main drawback of such an architecture is the difficulty of developing a common framework that can be used to express the potentially complex interactions between contexts. In a sense, Plutarch shifts the problem of developing a common network layer into the problem of developing a common negotiation framework, i.e. the shared mechanisms that will be used at the boundaries of contexts to translate protocols and data.

Here there is a clear room for autonomic networking principles where self-managed entities could autonomously negotiate how Plutarch's interstitial functions should be dynamically instanciated on-demand. Realistically, one could indeed not rely on humans to specify all the possible and future contexts interactions in a networking world where each operator is free to implement its own customized (and potentially private) protocol suite.

4.7 Turfnet

The TurfNet [69] architecture focuses on enabling communication among highly autonomous and heterogeneous network domains that may use very different internal protocols and addressing schemes. It concentrates its new internetworking mechanisms at the gateways, enabling easy integration of legacy end-user nodes. The architecture uses a global identifier namespace and does not require global addressing or a common internetworking protocol. Similar to many other proposals, it decouples locators from identifiers. The TurfNet namespace maps names into logical node identities and a second mapping translates node identities into node addresses that are suitable for network-layer data forwarding. The TurfNet architecture manages the global name and identifier spaces, whereas address spaces are local to each individual turf (and independently managed).

A key feature of the TurfNet architecture is network composition. Isolated, autonomous turfs can dynamically compose into new, larger internetworks that integrate the original turfs. The process of dynamic network composition supports the interconnection of heterogeneous networks, such as mobile and ad hoc networks, and IPv4 or IPv6 networks. Composed "super" networks manage this integration by abstracting away potential conflicts (e.g., overlapping address spaces) or heterogeneity issues (e.g., incompatible network protocols).

End-to-end communication across turf boundaries is not trivial, due to the potential heterogeneity of the individual networks. Turf nodes use node identifiers to address communication peers. Because locators are turf-local addresses, turf gateways generally need to translate between intra-turf address spaces and protocols. To establish the necessary translation state, TurfNet uses explicit node registrations and address lookups that configure paths across composed turfs. End-to-end communication between nodes can begin as soon as the lookup process completes. Data communication follows the path established by the registration and lookup operations. The specifics of inter-turf communication depend on the involved turfs: if they use the same address space and communication protocols, the gateway nodes can simply act similar to traditional routers and forward traffic; otherwise, turf gateways must also perform the necessary address and protocol translations.

Turf boundaries correlate with administrative boundaries, i.e., a single turf is owned and thus controlled by a single entity. When turfs compose into TurfNets, bilateral composition agreements with customer, provider or peering turfs determine how internetworking occurs and what kinds of services neighboring turfs

exchange. For example, within the bounds set by a composition agreement, a provider turf may control how it handles node registrations, lookups and transit of customer data traffic. The fact that individual turfs are independent from global addressing, network protocols or external services increases their autonomy, compared to Autonomous Systems in the Internet, for instance. Because the ability to operate autonomously is a prerequisite for self-organizing networks, TurfNet can be considered a stepping-stone towards autonomic internetworking.

4.8 Summary

At first sight, the various proposals reviewed in this chapter seem to cover a very broad range of alternative design principles and architectural tenets. However (and somehow surprisingly), they share a number of architectural concepts, i.e. an observation that supports the fact that a prominant part of the research community is embracing a number of design principles.

As already discussed at the end of the previous chapter, the identifier and locator split is a widely agreed concept. So is the explicit coexistence of independently managed addressing realms. In addition, the more recent proposals presented in this chapter ([62, 67, 69]) assume that independent addressing realms can also use distinct and potentially incompatible addressing schemes. If ANA embraces this design, a challenging task will be to design autonomic mechanisms that translate packets traveling from one addressing realm into an "incompatible" one.

Chapter 5

Active & Programmable Networking

The promise of additional flexibility and intelligence offered by Active Network research made it a playground of choice to develop prototypes of what could be autonomic networks in the future. We present some of these works that address the problems of self-managing, self-healing or self-optimizing systems in sections 5.4, 5.5 and 5.3 respectively.

This chapter will also review some of the proposed active platforms that presented interesting ideas to achieve self-configurability in section 5.2.

5.1 What Are Active Networks

In the traditional model of Internet, router software contains the whole logic to forward packets to the proper interface solely based on the *destination address* carried by packets and the content of the *routing table*. All the additional complexity (congestion control, error recovery) is handled by end systems. Nowadays this simple "store and forward" model is extended over and over to provide more security (firewalling and filtering rules), better guarantees (differentiated services through queueing and scheduling) and improved performance (explicit congestion notification, local packet caching and retransmission).

Breaking with that model, the DARPA "Active Networks" project [70] proposed that end-systems (and thus end-users) could replace that plethora of specialised protocols with *programs* injected into the network and processed by routers. This model defined an *execution environment* combined with an abstract operating system (NodeOS) [71] that would coordinate the evaluation of those programs. A management framework for the active networks DARPA program was also specifically developed and implemented [72]. Beside the base model sometimes referred to as "Strong Active Networks", a second approach – known as "Moderate AN" – suggests that active code is provisioned by the network operator himself. This

chapter considers the two approaches. Finally, note that earlier proposals related to programmable networks include [73] and [74].

5.1.1 Packets Carrying Programs

Amongst the proposed models, the *capsule* is a paradigm where each packet contains the complete program stating each step to be performed in order to forward the packet. A typical capsule will read some packet status, lookup for information left in the active router by other capsules and then issue a "forwarding table" lookup to the NodeOS to know on which interface it will go. When pushed to the extreme, the capsule could even choose its position in the interface's queues or completely handle its own forwarding table. Since it is fairly difficult to put such complex programs within data packets, the ANTS project [75] proposed that code should actually be downloaded separately and *cached* on the router. Each ANTS capsule only carries a *reference* of the code that should be used (computed from a MD5 hash of that code) for the purpose of locating the appropriate code in cache or to download it from the previous node.

On the other side, in the CANEs project [76], programmability is only offered at pre-defined *slots* that are present in a *generic* packet processing function. In essence, this is very close to the programming model of *netfilter* in Linux kernel.

The various active platforms proposed usually differ by the technique they use for transporting code, by the control they offer on the underlying node and the mechanisms they enforce to ensure proper operation of the node.

Running user-issued programs within the network has however raised a number of issues. While many approaches have been envisioned to make those programs safe (through interpretation or sandboxing, for instance), or give proof of their safety (known as proof-carrying code), it remains unlikely that we could afford such checks for every data flow the router should handle, and the deeper we go into the network, the more unlikely it becomes.

5.1.2 Programmable Switches

For the network operators point of view, giving user control on how the packet should be handled is perceived as a threat rather than a progress, since planning and provisioning bandwidth becomes almost impossible. However, the idea of having quickly deployable packet-processing functions on demand has received significant interest. The notion of *active services* encompasses those projects where extensible functionalities are available to the user by means of operator-provided *plugins*. The selection of those plugins, the subset of nodes where they could be installed and the downloading and installation procedure remains under the strict (automated) control of the network operator.

The way users specify which active services should be applied to their own

packets may vary from platform to platform, but the general trend is to opt for a web service where the users set up *overlay topologies* interconnecting sites that should benefit from the same service. The main drawback of this approach is that it is often hard to extend it to the interdomain, since it is unlikely that all the operators on an Internet path will agree on which plugins to provide, the very same way they do not agree on a global multicast or QoS infrastructure.

5.1.3 Deploying Active Networks

How could we test a new network paradigm that changes even the way forwarding is achieved? The idea of having a "native" packet format for active packets, replacing even the IP layer, and being exchanged only between active routers has been abandoned quite early, as well as the idea of using something else than IP for locating machines that were part of the active network.

Yet, it was necessary, for testing purpose, to have paths with active routers *within* the core network, so the A-Bone [77] project has been set up and offered NodeOS-compliant platforms for packets carrying the Active Network Encapsulation Protocol (ANEP) header.

Apart from testbed environments, however, it is unrealistic to expect every router to be active (and to support the execution environment a given active packet requires). Core routers typically handle millions of flows [78] and therefore need to be as stateless as possible. Moreover, technologies like MPLS are common within the network core, meaning that the packet might actually not even *see* core routers.

In several proposed architectures, only the *edges* of a domain are active and interconnected with legacy hardware. The main two proposed approaches to handle heterogeneity of active/legacy nodes within the network are *active option* and *overlays*. In the overlay approach, active nodes in the network are identified and connected to each other by means of *tunnels*, making the active packet appear as a legacy packet until the next active node receives it, and making the tunnel appear just as a point-to-point wire for the active nodes.

On the other side, the active option paradigm [79] suggests that active routers will be able to identify active packets and do what is required and that legacy router will handle them exactly the way they handle legacy packets. In this framework, it is up to the active protocol designer to ensure that her solution keeps working (or degrades gracefully) when the density of active nodes in the system decreases.

Both approaches have benefits and drawbacks. While overlays maintain the illusion of a fully-active network, they involve substantial building and maintenance overhead. Several works explore how this can be done in specific applications (such as YOID [80] in the case of multicast transmission or Chord and Pastry [81, 82] for distributed hash tables), but dealing with unplanned overlay requirements across different network operators remains unsolved so far and solutions sketched such

as X-Bone or OPUS [83, 84] require a quite heavy infrastructure for operating successfully.

Another challenge for overlays is to maintain the illusion that the tunnel has foreseeable characteristics while the underlying network might very well change the route tunnelled packets take, suffer from congestions due to "perpendicular" traffic, etc. A regular point-to-point wire will not reorder packets it carries; a tunnel could very well do, and it is not obvious to predict when it will do [78].

Several parameters distinguish overlay networks from 'real' networks, like the fact that link costs may change at any time (due to a change in the underlying topology), or the fact that there is usually no broadcast facility to discover peer routers. With the "active option" approach, no such infrastructure is required, but we might very well find no active router on a given path while a small deviation of the standard path might have found one.

Sivakumar et al. [78] highlights 4 questions to be answered by an active network architecture:

1. Where should active routers be placed in the network?
2. If not all routers are active, how can we abstract the state of 'non-active' portions of the path?
3. How do active routers discover and communicate with other active routers?
4. How do users discover third-party services?

5.1.4 Offloading Active Services

To some extent, many of the applications suggested by the active network community do not need to take place *on the routers*. They mainly need to be applied *within the network*. If an overlay topology is built, nothing actually requires that the machines that process active packets *are* the physical devices that forward the packets.

In many "high-performance active node" research, active packets are identified by the ingress line card and dispatched through the switch fabric to a "processing card", an additional hardware component that acts as a line card as far as the switch fabric is concerned, but which actually features a general-purpose processor capable of doing the active processing required and then re-inject the packet in the switch fabric once its actual destination (or queue level) has been decided.

The Tamanoir Active Node [85] extends this concept by suggesting the use of a processors cluster site connected to the border routers so that even heavy tasks (such as video flow transcoding) can be achieved at high rates.

However, Sivakumar et al. [78] claim this approach cannot effectively support many of the services that motivated the design of active networks in the first place.

It would be for instance more difficult for an "offloaded" active service to know the current state of the routers it monitors (e.g. what is the *instantaneous loss rate* of a wireless link, and should we provide more forwarding error code to increase the chance that the packet transmits successfully).

5.2 Active Platforms

5.2.1 ANTS

ANTS [75] is a capsule-oriented platform, where code used for packet processing has access to a limited set of primitives:

- environment access functions such as retrieving node address, channel properties, localtime.
- store and retrieve data from the protocol-specific *soft-store*.
- route capsules towards another node or deliver it to a local application.

Capsule code is organised into *protocols* (e.g. set of capsules that share code and have access to the same *soft-store* on a router). Both capsule and execution environments are written in JAVA, which gives portability, mobile code support, and a good substrate for the required safety. ANTS comes with its own code download mechanism, based on MD5 hashing of the required bytecode (carried by the capsules), so that the *previous active node* can be requested to transmit the code group needed for interpretation of a given capsule while maintaining the guarantee that the code that will operate on the capsule *is* the code expected to be used[1]. This mechanism has settled the base of self-managed code deployment for many subsequent works in the field of active networks.

An open-source distribution of ANTS running on top of the Java Active NodeOS [86] is available at university of Utah since 2001, which has led to a collection of ANTS-derivative works, some improving the performance while maintaining the programming model [85], others providing alternate code download mechanisms to keep services deployment under the control of the network operator [87]. Others integrate a self-configuring mechanism for partially active networks [88] on a platform that otherwise requires the testbed topology to be expressed through configuration files.

In the specifications, ANTS node could also host various *services* that capsules could connect to using the `findExtension` primitive. Such services could have featured a database lookup, or any other 'specialised' function, but it has - to our best knowledge - never been used or implemented.

[1] as long as we trust MD5 as a one-way hash function, and if we can trust the router operator to run a properly-implemented ANTS router.

Beyond the fact that it is unclear whether a heavy environment such as JAVA can be supported in a core router[2], the design of ANTS raises other issues.

First, each protocol has the option of storing data in the *soft store* (a key-based object storage). These data can be explicitly removed by the protocol or they can be reclaimed by the node after a pre-defined 'idle' timeout. Every time a specific item in the soft store is reused, the 'idle' timeout is restarted, which means a thin flow of capsule is sufficient to lock some memory on the node for arbitrarily long periods of time. To keep things running, ANTS will use per-context memory allocators, making sure than one protocol cannot consume the whole memory of the node. However, defining the appropriate limit while maximizing the number of running protocols is not a trivial task. This however implies that several 'contexts' may have to synchronise themselves and cooperate to handle a capsule, for instance when code for that capsule needs to be downloaded or when the capsule is delivered to an application, which results in additional context switching and inter-context communication overhead.

In addition to ease memory management, per-protocol store is a primary requirement to ensure that it is not possible to write a 'scanning' protocol that crawls the network, detecting what other protocols are running and modifying their state for malicious purposes: protocols only compete for resources but they do not interfere with each other states or capsules.

By a sophisticated combination of JAVA sandboxing features, appropriate design patterns for exposing information about the node without letting capsule code alter them, and thanks to the inherent type-safety and bounds-checking of the language, ANTS manages however to keep the node as a safe place for everyone – what alternate solutions based on native code cannot usually enforce unless the code has been compiled locally with a 'secure' compiler adding safety checks.

Yet, malicious code *may* be written in the ANTS toolkit. Because JAVA language provides no mechanism to guarantee code completion in bounded time constraint, even a small capsule can consume CPU resource indefinitely. In ANTS/Janos platform, this is achieved by *watchdog timers* that will ensure long-running forwarding routines are terminated. Once again, it is not obvious to define those watchdog timers so that they catch *only* misbehaving code. A more concerning issue is that, according to Hawblitzel et al. [90], it is unsafe to terminate runaway threads in the JVM[3].

A steady flow of capsules where each capsule executes a small action could very well be consuming the same amount of CPU resources as a sparse flow where each capsule asks for long operations and yet the later one would be terminated by the watchdog and the code would likely be tagged as misbehaving, preventing subsequent capsules with the same code to execute on the node.

[2] [89] reports 4,000,000 packets/sec (minimal packet size of 70 bytes at OC-48 wire speed) in the CISCO 12000 series. On the other hand, the "Click Modular Router" achieves 'only' 70,000 packets per seconds on PC hardware according to R. Morris et al. (SOSP'99)

[3] quoted from [91].

5.2.2 Protean

The PROgrammable TEchnology for Active Networks project [78] follows the 'extensible services' approach. A common packet classifier decides which *user network context* (UNC) should be applied to each packet. Those contexts contain a collection of *(event, handler)* bindings, customizing the processing that the packet will receive. Events include system events such as "incoming packet", "packet ready for transmission" or "packet dropped", but also triggering of periodic timers or custom events that other event handlers fire off. The handlers are kept in dynamically loadable kernel modules, identified via *unique service profiles* and optionally retrieved from other nodes using the dedicated SPINE infrastructure.

The protean switch is equipped with a compiler for a safe subset of C language (subC) that will produce native code safe for kernel-level operations out of source text downloaded from a nearby cache or directly from the source – a technique that is quite common in the "programmable switch" approach[4]. From a security perspective however, this might be easier to fool than ANTS's hashing scheme (e.g. nothing guarantees that the source code retrieved from a nearby cache will actually do what the end-user expects).

Unlike what happens in *ANTS*, it does not only define the node architecture, but also network services that are required to support operations of the nodes. Among other things, the PROTEAN framework suggests that only a portion of the nodes – the edge of the domain – might be programmable by the user, for scalability purposes. In addition, PROTEAN comes with its own hierarchical lookup service to retrieve the storage location of a code module based on compound names such as "edu.uiuc.crhc.timely.siva.md5cksum". Finally, PROTEAN offers a *link abstraction* to compute and provide estimated characteristics of tunnels between programmable nodes in terms of loss rate, delay and bandwidth, helping the services programmer build services that will work even when not all nodes are active.

The notion of *on-demand packet processing path* through components that can be chained to each other, detailed in those works, is most likely a basic feature that we could need at the core of an autonomic node in order to allow self-optimization (building a dedicated "fast path" for a given class of traffic). Specifically, the "guarded service chains" model presented by Ruf et al. [92] and its associated control interfaces seems to be a good candidate. Recent works even optimize further those custom paths by generating code for network processors or field prorammable gate arrays (FPGAs) from components and path descriptions.

5.2.3 PLANet / SwitchWare

Compared to most other projects that distribute the active code either via *out-of-band* mechanisms (e.g. active packets contain an identifier used to retrieve the

[4] the OKE Corral (IWAN'2002) uses a similar approach.

plugin either from a cache or from a code server [93]) or via *in-band* mechanisms (e.g. the code is present in the packet), PLANet uses an interesting mix of the two. The active packets contain scripts expressed in a safe language – PLAN [94]: A Packet Language for Active Networks – that will make use of out-of-band installed *active extensions* that are under the control of the network operator. The PLAN scripts therefore act as a "glue" for operator-offered services composition.

Safety and security in *PLAN* are achieved through the language design rather than from virtual machine properties. PLAN is a functional language whose expressibility is limited but that guarantees termination of programs. Active extensions can balance that limitation by offering more complex services.

One of the main drawbacks of PLAN is that the program representation can be quite long and that converting that representation into something that can be handled by the interpreter (e.g. parsing the code, unmarshalling packet-carried data) can be inefficient. SNAP (Safe Networking with Active Packets [95]) is another packet language defined in the context of the *SwitchWare* project of University of Pennsylvania and designed to fix that problem. While PLAN uses a high-level functional language that has to be converted between network representation and executable representation at each node, SNAP bytecode is executable *in situ* in the packet buffer, combining a compact representation with high speed interpretation.

For a programmer's point of view, SNAP splits the packet between "stack" and "heap", where the heap contains for instance the data arrays for the active extensions. The authors of SNAP have proven that each instruction either grows the stack by one item, or shrinks the stack. Moreover, SNAP can only take *forward* jumps, which means that evaluation time is now a *linear* function of the code size (in PLAN, the execution time could be bounded by a function of the code size as well, but it could be *exponential*). Finally, when a new packet is sent, its initial *resource counter* value is subtracted.

As a result, if we consider a set of packets $N = P_1, \ldots, P_n$ in the network, the sum of resource counters for each packet is a *termination function* for the computation expressed by those packets, and thus bounds the overall amount of processing required on these packets. The following properties are demonstrated on SNAP packets:

1. On any node, processing a packet p only takes $O(|p|)$[5].
2. On any node, processing a packet p only requires $O(|p|)$ memory.
3. The overall network bandwidth consumed by a packet p is at most $O(n|p|)$ where n is the resource bound by the packet at its creation.

An exhaustive list of SNAP bytecode is given in Jonathan T. Moore's dissertation "Practical Active Packets" [96]. Among the interesting derivative works, a compiler to translate PLAN into SNAP is presented in [97] and a just-in-time

[5] $|p|$ is the length of packet p in bytes.

compiler of SNAP bytecode on the PowerNP network processor is presented in [98].

5.2.4 Ephemeral State Processing

Ephemeral State Processing (ESP) [99] from University of Kentucky provides an open interface to deposit, retrieve and process small pieces of data in routing elements while keeping IP's inherent properties such as robustness, anonymity, generality, and processing efficiency. ESP does not propose a *replacement* for Internet Protocol, but rather comes as an extension to the existing network layer that should provide the bare minimum flexibility required to help user applications build more capable solutions such as reliable multicasting [100].

ESP defines a small set of generic operations that manipulate the state stored on the routers. Each *ESP packet* processed by an ESP router invokes one of these instructions over a specific set of data and may lead to modification of the ESP packet's content, modification of the router's stored data and a decision on whether the packet should be forwarded or discarded.

The header of ESP packets contains a numerical operation code that identifies the computation to be applied (5 generic operations are available). The ESP packet also contains the operands to be applied for that specific computation, such as the immediate values involved in the computation and the specific entry in the router's store (identified by a "tag") to be manipulated.

The tag is a 64-bit integer that acts as a variable name on that router. For a short period after that packet has been forwarded (and that is why the store is said to be *ephemeral*), any packet that reuses the same tag can retrieve the value computed by the first packet, update it, etc. When that period is over (typically 10 seconds), the binding between the tag and its associated data is removed.

The "ephemeral store" mechanism offers nice advantages over a typical "soft-state" store, most notably it is possible to bound the amount of memory that will be required to process a given flow of ESP packets. Moreover, it requires no access control or end-user authentification, which makes it potentially implementable at higher rates than what ANTS and related proposals can achieve. The tags used for information storage are simply random numbers picked up by the end-systems (and optionally exchanged with peer end-systems through secured channels).

WASP (World-friendly Active packets for ephemeral State Processing [101]) is a derivative work in progress that extends ESP routers with a small interpreted language to replace built-in "generic operations" and allows a larger application range to ephemeral store. Among other things, it can easily gather information along a path between two nodes, measure flow jitter or implement small control protocols. The interpreter is however under tight control to ensure that WASP packets do not affect routers or legacy network operations.

Note that, by design, both ESP and WASP are more "zero-configuring" than "self-configuring", which makes them particularily suited to the core of an extensible node to provide the "minimum services for maximal extensibility".

5.3 Self-Optimization, the Active Networks Way

An important share of active and programmable network research focuses on optimizing the use of network resources for a given application. The fact that the network is programmable is indeed a nice opportunity to benchmark a new optimization technique (which would otherwise have required an arguable modification of the core router), but also the increased packet processing cost would be meaningless if it could not achieve a sensible performance improvement. Depending on the application, optimization may involve different mechanisms and therefore imposes different requirements on the architecture [78]:

routing the service allows the modification of the path taken by packets, either to provide better quality of service, multipath routing for resilience, or mobility support.

differentiation the service provides differential packet processing by selecting scheduling, dropping priority, etc. to be applied.

data manipulation services like transcoding, compression, decompression, encryption, decryption operate directly on the application data flow, modifying even the payload (and thus potentially the number of packets and their size) of the flow.

data forwarding services may alter the end-to-end communications by caching, snooping and retransmitting data, but they do not "produce" new data themselves.

5.3.1 Active Caches

In several client-server applications, including the Web, user-perceived performance benefits from the presence of *caches* within the network, installed on intermediate systems (usually *proxies*). Since only a small fraction of available content (i.e. the *popular* content) accounts for most of the traffic, keeping a copy of popular items in a cache located nearby clients usually saves both server bandwidth and request latency. Arlitt [102] estimates that only 0.3 to 2.1 percent of server-offered content is requested, and that it is frequent that only 10% of the content accounts for up to 95% of all requests, which explains the performance achieved by hierarchical distributed caches where a local cache miss can be resolved at sibling or parent caches [103].

Using active networking, it is also possible to get rid of the management hassle inherent to hierarchical caches. Rather than having a few proxy caches with large storage capabilities, Bhattacharjee et al. [104] suggest that all the nodes should be involved and provide room for a few objects. Those caching routers use a self-organising policy to know what router should cache what object. The "lookaround" policy suggests to reserve a small portion of the storage capacity to keep *pointers* to objects stored in neighbouring nodes, a scheme that we find back in recent peer-to-peer distributed storage research such as the OceanStore project.

5.3.2 Multimedia Flow Transcoder

A problem that arises when one tries to send a multimedia flow to a collection of receivers is that all the receivers might not have the same capabilities: some may lack the CPU power to decode a more complex encoding, others may not have the required bandwidth to support the whole flow. Others, again, might be hardcoded to support some encoding and know nothing about the newer technologies the sender is streaming.

The idea of active video/audio transcoders [105] was to detect these situations and to install transparent repeaters in the network that would help the source by providing streams with lower qualities (and lower bandwidth) or transcode the whole stream to a newer encoding (such as MPEG to H.323 conversion and back). While this is a good example of CPU-intensive feature that could happen in the core network, it is quite arguable (and has been often criticised) whether it is a good thing to have it *within* the network rather than at the source.

In many scenarios, active *transcoding* could be efficiently replaced by active *dropping*, making sure that, if not enough resource is available to carry the whole stream, then at least the most interesting parts of the stream are indeed received. Early works with MPEG [106] have shown that dropping entire group of pictures or dropping B-frames when I-frames's fate is uncertain can lead to significant improvement of the received signal. With layered video streaming in general, and MPEG2000 specifically, it gets even easier to keep useful information with a reduced bitrate since the encoding is organised in such a way that it is sufficient to drop packets that belong to one layer to get smooth degradation of stream quality [107].

5.3.3 Active Congestion Control and Cognitive Packets

The works on in-network video transcoding (and content adaptation in general) can be seen as a specific response to the general problem of network congestion handling. On some topologies, a cheaper alternative to content transcoding is obviously to reroute the flows to avoid congestions or even to introduce in-network "traffic shaper" to reduce flow rate in routers nearby the congestion. *Active Congestion Control* (ACC) framework proposed by T. Faber, interesting for the way it

suggests to use programmability to combine the different approaches, and *Active Queue Management*, from University of Kansas are examples of the attention this subject has received during the "DARPA years".

On the other side, *Cognitive Packet Networks* (CPN, now also known as "self-aware networks") is an intriguing challenger that aims to solve the general problem of QoS and congestion management in the network by a complete replacement of the network protocol. CPN [108] uses neural network in each node to learn and reinforce the best next hop for each source/destination pairs. Note that only a small portion of the packets are *smart* in CPN – that is, look for new routes and reward/punish nodes for their decisions – while the actual traffic is carried by "dumb" packets that simply follow indications stored in the nodes.

Over the time, however, the cognitive packets have dropped their "executable code" portion [109] and no longer require node programmability. A prototype implementation of the RNN algorithm on FPGA achieving nearly 600Mbps has been recently presented and sounds promising even if the accuracy of its approximation remains vague.

5.3.4 Peer-to-peer Optimization with Active Networks

The recent explosion of peer-to-peer traffic is a double challenge for network operators. First because P2P systems exhibit properties that strongly differ from well-studied applications for which ISPs networks have been optimized, and second because most of the P2P applications quickly react to new detection techniques to remain as undetectable as possible. Once a P2P overlay is detected by an operator, however, it becomes possible to optimize the network performance by shaping or rerouting P2P traffic away from interactive sessions or even by redirecting connections to a local application-aware proxy.

As detailed by Dedinski et al. [110], there are several aspects of this procedure that can greatly benefit from a programmable network, like the aggregation of micro-flow properties (inter-packet timings, for instance) or the building of similarity classes. When needed, an active crawler for a given P2P protocol can be started on a node to collect or validate assumptions, and active routers can exchange information to build a high-level topology of a given application, therefore easing the identification of client/server, hierarchical or peer-to-peer applications.

5.4 Self-Management, the Active Networks Way

5.4.1 Active Monitoring and Management

Network management remains a field where active networks are highly appealing. Indeed, collecting information from a large pool of machines, identifying recurring

events or coordinating actions is indeed still impractical with standardised tools such as SNMP. Active management entities could receive complete programs that would watch a specific combination of events (where traditional management systems only allow to install triggers for a single event in best cases), and exchange information with peer agents in other monitored systems or take immediate response as chosen by the management station. Another advantage of active management is that it allows the system to react even when the management station is offline.

Some other platforms such as SmartPackets or SNAP [111, 91] also attempt to propose active network solution for the "management agents" paradigm, coupling a safe language with an interface to node's Management Information Base (MIB). Mobile Agents platforms are indeed more interesting if one can be sure that a misbehaving agent (due to a programming error, for instance), will not be able to remain indefinitely in the network once unleashed. Resource-bound design of active network can help here design agents that can only live for a few dozen of hops and will then have to return to the *network operations centre* where the information they gathered is analysed and another replacement agent might be released. With that paradigm, however, we lose a bit of flexibility for safety, since it means agents cannot "settle in" at a specific node to keep monitoring it.

Other works such as HABA/CAA/CAN [87, 112] present an interesting way of monitoring their local resource and reporting them. A domain-local manager is responsible for aggregation and dissemination of the network monitoring in order to perform e.g. quality of service reservation on an interdomain path.

Some active networks proposals even specifically target network monitoring, like the Self-Configuring Active Network Monitoring (SCAN) platform proposed by University of Southern California [113], which proposes an interesting mechanism to cover the network with *surveyors* that automatically partition the network into monitoring domains, and suggests a framework to infer/exploit information.

The actual impact of active networking and programmable networks research on autonomic networks will of course depend on the amount of *programmability* we want on the autonomic nodes. However, while we could probably build a self-aware, self-optimizing and self-managing node (or even network) with a "fixed" program, it could remain interesting to deploy specific code component to enforce a decision made by the autonomic management software, especially when that component is to be executed on sophisticated programmable devices such as network processors or field programmable gate arrays (FPGAs).

5.4.2 Service Discovery using Active Routers

One of the basic property required to build autonomic networks in general (and self-configuring domains in particular) is the ability to identify nodes that are capable of hosting a given high-level *service* – such as name resolution or topology/measurements aggregation – and to advertise their location to the rest of the network.

An important amount of work addresses such discovery problems over the Internet, and beyond DHCP or DNS, the Zeroconf IETF working group has for instance proposed DNS-Service Discovery (DNS-SD) [114] and its multicast variant (used in MacOS X "Bonjour"). When the 'locality' goes beyond a simple LAN, but remains topologically close to the client, the *expanding ring search* is one of the most common techniques used. The key idea is to make use of multicast support to send service location requests towards a multicast address that all potential service providers will be listening to.

On the opposite side of the spectrum, global services discovery will locate an item *wherever it stands*. This is the kind of abstraction offered by DNS, and similar hierarchical databases are proposed in active networks literature, such as Spine, the network infrastructure protocol used with the *protean* active network. Distributed Hash Tables [81, 82] may also offer an interesting alternative when information cannot be easily split along a hierarchy.

As detailed in [115], we may also need that a high-level service operates on a data flow that simply "crosses" a transit domain (such as content caching). In today's Internet, such services are hard to offer because end-systems typically ignore the transit domains their packets will cross. In an autonomic network, we could take advantage of the components' key/value store to allow detection of available services at the time of connection establishment. The discovery then takes place as a two-phase process: servers A and B first flood the domain with active packets advertising their presence, avoiding to re-install an advertisement in a router that already contains a better one (e.g. advertising a closer or less loaded service provider). A source S can then use another active packet to record those advertisements as a list of provider P and branch point X information.

This approach allows the network operator to remain in control of the additional load generated by service lookups in his domain by deciding the refresh rate of advertisements. However, since advertisement is limited to the domain that hosts the service provider, the end-systems still need to have an initial destination and will only find providers from domains that lead to that destination. In other words, unlike what a global *anycast* [116] service could offer, you cannot "get the closest news aggregator" here, but you can "get the closest news aggregator between me and slashdot".

5.5 Self-Healing in Active Networks

Several recent active applications proposals are targeted at Distributed Denial of Service or worm propagation mitigation and fight-back [117]. The ability of running downloaded code (from secure sources) that tune packet filters according to a new attack signature or to exchange and collect information from thousands of sites and correlate them to identify the attack pattern stems for programmable equipments very similar to those proposed by active networks, even if here the end-user is not invited to make use of active code.

As detailed in [118], programmable networks are almost mandatory to build a network resilient to flash crowd and DDoS attacks, as new patterns of attack may appear at any time. Be it for detection modules that depend on application-layer protocols, for appropriate pushback protocols (to throttle the offending traffic) or for network reorganization (to keep cross-traffic running), attempting to pre-program a resilience solution sounds like a futile exercice.

With the recent development of programmable network hardware such as network processors or the Field-Programmable Port Extender (FPX), one can now design pattern-detection engines that enforce malware removal *in the access network* rather than on the end-systems [119], and with proper collaborations of local security monitors, we might even have a chance to contain appearing worms before they cause excessive damage [120].

5.6 Summary

As we have seen throughout this chapter, several proposals made in active and programmable networking research already address the issue of self-* properties. It is clear, however, that these works are just providing initial hints towards an autonomic architecture and that there is still much to do to achieve self-management. In particular, active network research has brought little insight so far on the dynamics, including the autonomics of mobile code and of the network architecture in general.

Moreover, the existence of solutions to some of the self-* issues with an active/programmable network technology does not imply that we *need* active network in an autonomic architecture in the first place.

This chapter also shows that "active networking" is not just one solution, but rather a large spectrum of solutions that vary in how programmable the network is. User-submitted capsules are just one end of that spectrum, and maybe the one we could easily dismiss in the ANA blueprint: we could hardly call "autonomic" a network whose behaviour would be entirely defined by the code sent from end-systems.

On the other side, as stated in Yamamoto et al. [121], it is hard to imagine how we could achieve *automatic adaptation* of the architecture if components are not capable of modifying their behaviour (e.g. by self-modifying their software).

Considering the case of *cognitive packets*, however, it seems too early to state whether programmability is a primary requirement for ANA, or just a convenient way to reach our goal.

Anyway, even if we do not necessarily intend to make the ANA architecture an active/programmable network, some of the mechanisms developed for the purpose of active networking can be interesting basic blocks to build it.

Through our review of active platforms, we tried to highlight such potential building blocks, such as ephemeral storage, automatic code deployment and dynamic services composition.

Each active platform of course comes with its own limitations, performance penalties and security concerns, so that the ANA designers should still carefully evaluate pros and cons of a given mechanism before they start building on it.

Chapter 6

Discussion and conclusion

As clearly stated in the introduction, the goal of this document is mainly to review and examine high-level definitions of network architectures. However while high-level design principles and abstractions allow network designers to clarify and organise concepts into a coherent framework, the aim of a network architecture is to be implemented and deployed. In this chapter, we briefly remind that various research fields will be examined when designing and implementing ANA. We will also check the progress of related projects in the field, and we will use state-of-the-art techniques to validate and evaluate ANA. We close this document with a short discussion on the current state of mind of the research community on the design of clean-slate network architectures.

6.1 Related research fields

A critical challenge when designing a network architecture is to turn high-level principles into concrete protocols, mechanisms, packet and data formats, etc, while strictly complying with the abstract definitions and objectives of the network architecture. This step is where the designers and implementors should carefully import state-of-the-art techniques and protocols from related research fields. These include for example work in network security, management and monitoring, fault-tolerance, and applications (grid computing and peer-to-peer networking to cite just a few).

This separation of concerns was achieved in ANA by separating the work into several work packages. Work Package 1 will produce the high-level "blueprint" of the network architecture, which will also include more detailed specifications such as APIs and descriptions of low-level mechanisms (e.g. basic communication setup). Concrete implementations will be designed and developed by Work Packages 2 and 3, and integrated into a prototype by Work Package 4. Hence it is the task of these 3 work packages to develop specific functionalities in each related research field. WP1 is mainly interested by high-level principles and this is reflected

by the content of this document.

6.2 Other related projects

We here briefly want to mention that ANA is not the only project interested by autonomic networking. The EU initiative in Situated and Autonomic Communications (SAC) approved four projects (including ANA). Haggle [122] is interested in enabling communications in the presence of intermittent network connectivity, Bionets [123] focuses on biologically inspired mechanisms to achieve autonomic behavior, and Cascadas [124] aims at developing a framework for the deployment of a new generation of autonomic services.

Other projects, while not explicitly using the term 'autonomic', have similar interests to the ANA project. The objective of the Ambient Networks [125] is to enable seamless interoperation between heterogeneous internetworks. The BISON project [126] explored the use of ideas derived from complex adaptive systems to enable the construction of robust and self-organizing information systems. In the US, the Internet2 [127] project is building a new nationwide research network which for example incorporates the most advanced optical technologies.

During the course of the ANA project, and especially during the development phase taking place in work packages 2, 3, and 4, we will ensure that we carefully consider the most prominant outcomes of related projects.

6.3 Methodology and testbed

Over the last thirty years, various techniques have been used in order to design network architectures and validate their operation. At the design stage, formal methods can be used to validate algorithms and protocols. For example, such techniques include formal specification languages (e.g. ISO LOTOS), verification languages and tools (e.g. Promela/SPIN), and computational techniques (e.g. network calculus). In addition, simulation softwares and emulation platforms can be used in order to study more complex protocols that have many interactions. Popular tools include NS2, OPNET, GloMoSim, Emulab, and many others. During the design of ANA, we will use some of these techniques to assess our autonomic architecture.

Designing a network architecture of course also implies that a prototype must be tested and validated before one can claim that the design objectives were fulfilled. While small-scale testbeds are easy to build, larger testbeds require a high management overhead. To overcome this overhead, virtual testbeds such as PlanetLab have been developed. They allow to run multiple experiments on a generic substrate that ensures that each test/project operates in a virtual environment which is not influenced by other experiments.

With broader objectives, the GENI initiative [128] in the US and the (currently discussed) ENGINE proposal in Europe advocate the need for a generic world-wide research infrastructure upon which next generation network architectures could be built and tested. GENI is a very ambitious, forward-looking, and challenging initiative which could be the trigger towards innovative network architectures. Clearly ANA and GENI have similar goals, the most important one being to design a flexible system on top of which different (and potentially concurrent) network protocols and architectures can run. While GENI puts its emphasis on the physical deployment of such a network, ANA focuses on the "autonomic" behavior of the system: this is where ANA will provide the majority of its innovations.

6.4 "The time is right"

As stated many times in this document, the research community widely agrees that the evolution of the core of the Internet has been limited over the last twenty years. This does not question the immense deployment success of the Internet and the very large diversity of applications: the concern is that variability and innovation is restricted to the application layer. Emerging networking paradigms are over-stretching the design principles of the Internet architecture and a recurrent topic in the literature is whether these core principles should be relaxed in order to allow unconstrained innovations to appear. The recent renewed interest of the research community to design *clean-slate* network architectures indicates that radical architectural changes are envisioned. In a sense, "the time is right" to propose alternative network architectures (see e.g. [129]).

For example, the design of the current Internet architecture is based on the so-called "hourglass" model that promotes a unique and homogenous network layer. While this design allowed to easily interconnect distinct networks, it now also constrains the operation of emerging networking paradigms such as ad hoc and sensor networks: running a full IP stack on a sensor with limited processing power and memory might be very inefficient. In addition, more 'exotic' communication paradigms such as the ones used in delay-tolerant networks poorly fit in the Internet communication model of end-to-end connectivity, best-effort packet forwarding and highly reactive congestion control. As shown in the previous chapters, a large part of the research community seems to admit these limitations and many proponents now promote a heterogeneous networking model that does not rely on a unique and global network layer.

As already discussed in the previous chapters, (recent and older) papers in the literature are converging towards a number of networking principles. The identifier and locator split is widely approved, and so is the explicit coexistence of independently managed addressing realms (i.e. today's NATed domains). The idea that different addressing realms can also use different addressing schemes is also widely accepted, but while some argue that a new global layer should federate the network protocols (as IP federated link-layer protocols), others call

for the deployment of generic translation mechanisms along with ad hoc schemes to lookup and locate destinations and then forward packets. There seems to be more divergence whether the current design in layers should be maintained, or whether a more relaxed framework should replace layers (e.g. role-based architecture).

In addition to purely architectural and technical considerations, a new network architecture (yet an autonomic one) should not ignore human- and business-driven conflicts (i.e. Clark's tussles [28]). Modern networking is not just about protocols, algorithms, and packet formats: complex forces are competing and different stakeholders have opposite interests. A central theme in autonomic communications is that network entities can be steered by high-level objectives, trying to find an equilibrium between possibly adverse and contradictory goals. This "high-level steering" of an autonomic network will also change the way network management is performed. In the current Internet, heavy management efforts are overloading network operators which have to manually configure and tune networking systems and protocols. Here the envisioned self-* properties of autonomic networking are expected to allow networks to run with minimum human intervention.

Clearly, the ANA project was launched at the right time. The research community is currently receptive to alternative network designs and in the next few years we will witness the birth of many new network architectures. During the course of this project, we have to make sure that our architecture will differenciate itself from other proposals, thanks to the autonomic properties of ANA and to the deployment of a demonstrator.

Bibliography

[1] Autonomic Network Architecture. Integrated Project FP6-IST-27489. Sixth Framework Programme - Situated and Autonomic Communications (SAC), October 2005.

[2] H. Zimmermann. OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection. *IEEE Transactions on Communications*, 28(4):425–432, April 1980.

[3] S. Schmid, M. Sifalakis, and D. Hutchinson. Towards Autonomic Networks. In *Proceedings of the 3rd Workshop on Autonomic Communication (WAC 2006)*, September 2006. Paris, France.

[4] J. Kephart and D. Chess. The Vision of Autonomic Computing. *IEEE Computer*, 36(1):41–50, January 2003.

[5] M. Smirnov. Autonomic Communication - Research Agenda for a New Communication Paradigm, June 2004. Available at http://www.autonomic-communication.org/publications/doc/WP_v02.pdf.

[6] Autonomic Communications Forum. See http://www.autonomic-communication.org/.

[7] J. Saltzer, D. Reed, and D. Clark. End-To-End Arguments in System Design. *ACM Transactions on Computer Systems*, 2(4):277–288, November 1984.

[8] D. Clark. The Design Philosophy of the DARPA Internet Protocol. In *Proceedings of ACM SIGCOMM 1988*, August 1988. Stanford, CA, USA.

[9] D. Isenberg. Rise of the Stupid Network, May 1997. Released onto the Internet by AT&T in June 1997.

[10] J. McQuillan and D. Walden. The ARPA Network Design Decisions. *Computer Networks*, 1(5):243–289, August 1977.

[11] L. Kleinrock and F. Kamoun. Hierarchical Routing for Large Networks: Performance Evaluation and Optimization. *Computer Networks*, 1(3):155–174, January 1977.

[12] E. Gerich. RFC-1366 - Guidelines for Management of IP Address Space, October 1992.

[13] E. Gerich. RFC-1466 - Guidelines for Management of IP Address Space, May 1993.

[14] V. Fuller, T. Li, J. Yu, and K. Varadhan. RFC-1338 - Supernetting: an Address Assignment and Aggregation Strategy, June 1992.

[15] V. Fuller, T. Li, J. Yu, and K. Varadhan. RFC-1519 - Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy, September 1993.

[16] IAB-IESG. RFC-3177 - IAB/IESG Recommendations on IPv6 Address Allocations to Sites, September 2001.

[17] T. Narten, G. Huston, and L. Roberts. Internet Draft - IPv6 Address Allocation to End Sites - draft-narten-ipv6-3177bis-48boundary-02.txt (work in progress), June 2006.

[18] J. Shoch. Inter-network naming, addressing, and routing. In *Proceedings of 17th IEEE Conference on Computer Communication Networks*, pages 72–79, 1978. Washington, D.C., USA.

[19] J. Postel. RFC-791 - Internet Protocol, September 1981.

[20] B. Hauzeur. A Model for Naming, Addressing, and Routing. *ACM Transactions on Office Information Systems*, 4(4):293–311, October 1986.

[21] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed Diffusion for Wireless Sensor Networking. *IEEE/ACM Transactions on Networking*, 11(1):2–16, February 2003.

[22] A. Carzaniga, M.J. Rutherford, and A.L. Wolf. A Routing Scheme for Content-Based Networking. In *Proceedings of IEEE INFOCOM 2004*, March 2004. Hong Kong, China.

[23] L. Poutievski, K. Calvert, and J. Griffioen. Routing Without Addresses. In *12th IEEE International Conference on Network Protocols (ICNP)*, October 2004. Berlin, Germany.

[24] P. Mockapetris. RFC-882 - Domain names - Concept and Facilities, November 1983.

[25] P. Mockapetris. RFC-883 - Domain names - Implementation and Specification, November 1983.

[26] P. Mockapetris and K. Dunlap. Development of the Domain Name System. In *Proceedings of ACM SIGCOMM 1988*, August 1988. Stanford, CA, USA.

[27] M. Blumenthal and D. Clark. Rethinking the design of the Internet: The end to end arguments vs. the brave new world. *ACM Transactions on Internet Technology*, 1(1):70–109, August 2001.

[28] D. Clark, K. Sollins, J. Wroclawski, and R. Braden. Tussle in Cyberspace: Defining Tomorrow's Internet. In *Proceedings of ACM SIGCOMM 2002*, August 2002. Pittsburgh, USA.

[29] P. Molinero-Fernandez, N. McKeown, and H. Zhang. Is IP going to take over the world (of communications)? In *Proceedings of First ACM Workshop on Hot Topics in Networks (HotNets-I)*, October 2002. Princeton, NJ, USA.

[30] B. Carpenter. RFC-2775 - Internet Transparency, February 2000.

[31] D. Clark, C. Partridge, J. Ramming, and J. Wroclawski. A Knowledge Plane for the Internet. In *Proceedings of ACM SIGCOMM 2003*, August 2003. Karlsruhe, Germany.

[32] D. Clark, K. Sollins, J. Wroclawski, and T. Faber. Addressing Reality: An Architectural Response to Real-World Demands on the Evolving Internet. In *Proceedings of ACM SIGCOMM Workshop on Future Directions in Network Architecture*, August 2003. Karlsruhe, Germany.

[33] D. Reed, J. Saltzer, and D. Clark. Active Networking and End-to-End Arguments. *IEEE Network*, 12(3):67–71, May/June 1998.

[34] L. Peterson, S. Shenker, and J. Turner. Overcoming the Internet Impasse through Virtualization. In *Proceedings of Third ACM Workshop on Hot Topics in Networks (HotNets-III)*, November 2004. San Diego, CA, USA.

[35] S. Ratnasamy, S. Shenker, and S. McCanne. Towards an Evolvable Internet Architecture. In *Proceedings of ACM SIGCOMM 2005*, August 2005. Philadelphia, USA.

[36] A. Avizienis, J.-C. Laprie, and B. Randell. Fundamental Concepts of Fault Tolerance. In *Proceedings of the 12th IEEE International Symposium on Fault-Tolerant Computing (FTCS-12)*, pages 3–38, June 1982.

[37] J. Sterbenz and al. Survivable mobile wireless networks: issues, challenges, and research directions. In *Proceedings of the 3rd ACM Workshop on Wireless Security (WiSe'02)*, September 2002. Atlanta, GA, USA.

[38] F. Christian. Understanding Fault-Tolerant Distributed Systems. *Communications of the ACM*, 34(2):56–78, February 1991.

[39] R. Krishnan, J. Sterbenz, W. Eddy, C. Partridge, and M. Allman. Explicit Transport Error Notification (ETEN) for Error-Prone Wireless and Satellite Networks. *Computer Networks*, 46(3):343–362, October 2004.

[40] V. Raisinghani and S. Iyer. Cross layer feedback architecture for mobile device protocol stacks. *IEEE Communications Magazine*, 44(1):85–92, January 2006.

[41] R. Winter, J. Schiller, N. Nikaein, and C. Bonnet. CrossTalk: Cross-Layer Decision Support Based on Global Knowledge. *IEEE Communications Magazine*, 44(1):93–99, January 2006.

[42] V. Kawadia and P. Kumar. A Cautionary Perspective on Cross Layer Design. *IEEE Wireless Communications Magazine*, 12(1):3–11, February 2005.

[43] Z. Turanyi, A. Valko, and A. Campbell. 4+4: An Architecture for Evolving the Internet Address Space Back Toward Transparency. *Computer Communication Review*, 33(5):43–54, October 2003.

[44] K. Tsuchiya, H. Higushi, and Y. Atarashi. RFC-2767 - Dual Stack Hosts using the "Bump-In-the-Stack" Technique (BIS), February 2000.

[45] B. Ford. Unmanaged Internet Protocol. In *Proceedings of Second ACM Workshop on Hot Topics in Networks (HotNets-II)*, November 2003. Cambridge, MA, USA.

[46] R. Moskowitz and P. Nikander. RFC-4423 - Host Identity Protocol (HIP) Architecture, May 2006.

[47] R. Moskowitz, P. Nikander, P. Jokela, and T. Henderson. Internet Draft - Host Identity Protocol, draft-ietf-hip-base-06.txt, June 2006.

[48] H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, and M. Walfish. A Layered Naming Architecture for the Internet. In *Proceedings of ACM SIGCOMM 2004*, September 2004. Portland, USA.

[49] J. Saltzer. On the naming and binding of network destinations. *Local Computer Networks*, pages 311–317, 1982. Edited by P. Ravasio et al., North-Holland Publishing Company, Amsterdam.

[50] M. Handley and A. Greenhalgh. Steps Towards a DoS-resistant Internet Architecture. In *Proceedings of ACM SIGCOMM Workshop on Future Directions in Network Architecture*, September 2004. Portland, OR, USA.

[51] H. Ballani, Y. Chawathe, S. Ratnasamy, T. Roscoe, and S. Shenker. Off by Default! In *Proceedings of Fourth ACM Workshop on Hot Topics in Networks (HotNets-IV)*, November 2005. College Park, USA.

[52] R. Mahajan, S. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling High Bandwidth Aggregates in the Network. *ACM Computer Communication Review*, 32(3):62–73, July 2002.

[53] I. Stoica, D. Atkins, S. Zhuang, S. Shenker, and S. Surana. Internet Indirection Infrastructure. In *Proceedings of ACM SIGCOMM 2002*, April 2002. Pittsburg, USA.

[54] R. Gold, P. Gunningberg, and C. Tschudin. A Virtualized Link Layer with Support for Indirection. In *Proceedings of ACM SIGCOMM Workshop on Future Directions in Network Architecture (FDNA2004)*, September 2004. Portland, USA.

[55] C. Tschudin and R. Gold. Network Pointers. In *Proceedings of First ACM Workshop on Hot Topics in Networks (HotNets-I)*, October 2002. Princeton, NJ, USA.

[56] P. Francis. A Near-Term Architecture for Deploying Pip. *IEEE Network*, 7(3):30–37, May 1993.

[57] I. Castineyra, N. Chiappa, and M. Steenstrup. RFC-1992 - The Nimrod Routing Architecture, August 1996.

[58] D. Cheriton and M. Gritter. TRIAD: A New Next-Generation Internet Architecture, July 2000. Stanford Computer Science Technical Report.

[59] Internet Systems Consortium. Domain Survey http://www.isc.org/ops/ds/, January 2006.

[60] P. Francis and R. Gummadi. IPNL: A NAT-Extended Internet Architecture. In *Proceedings of ACM SIGCOMM'01*, pages 69–80, August 2001. San Diego, CA, USA.

[61] The NewArch consortium. New Arch: Future Generation Internet Architecture (Final Technical Report), 2003. Available at http://www.isi.edu/newarch/.

[62] D. Clark, R. Braden, A. Falk, and V. Pingali. FARA: Reorganizing the Addressing Architecture. In *Proceedings of ACM SIGCOMM Workshop on Future Directions in Network Architecture*, August 2003. Karlsruhe, Germany.

[63] R. Braden, T. Faber, and M. Handley. From Protocol Stack to Protocol Heap – Role-Based Architecture. In *Proceedings of First ACM Workshop on Hot Topics in Networks (HotNets-I)*, October 2002. Princeton, NJ, USA.

[64] N. Hutchinson and L. Peterson. The x-Kernel: An Architecture for Implementing Network Protocols. *IEEE Tran. on Software Eng.*, 17(1):64–76, January 1991.

[65] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. Kaashoek. The Click modular router. *ACM Tran. on Computer Sys.*, 18(3):263–297, August 2000.

[66] X. Yang. NIRA: A New Internet Routing Architecture. In *Proceedings of ACM SIGCOMM Workshop on Future Directions in Network Architecture*, August 2003. Karlsruhe, Germany.

[67] J. Crowcroft, S. Hand, R. Mortier, T. Roscoe, and A. Warfield. Plutarch: an argument for network pluralism. In *Proceedings of ACM SIGCOMM Workshop on Future Directions in Network Architecture*, August 2003. Karlsruhe, Germany.

[68] C. Bennet. The Yellow Book Transport Service: Prinsiples and Status. IEN 155. Internet Engineering Task Force., August 1980.

[69] S. Schmid, L. Eggert, M. Brunner, and J. Quittek. TurfNet: An Architecture for Dynamically Composable Networks. In *Proceedings of 1st IFIP TC6 WG6.6 Workshop on Autonomic Communication (WAC 2004)*, October 2004. Berlin, Germany.

[70] K. Calvert, S. Bhatacharjee, E. Zegura, and J. Sterbenz. Directions in Active Networks. *IEEE Communications*, 36(10):72–78, October 1998.

[71] L. Peterson (Editor). NodeOS Interface Specification, 1999. DARPA AN NodeOS Working Group Draft.

[72] A. Jackson, J. Sterbenz, M. Condell, and R. Hain. Active Monitoring and Control: The SENCOMM Architecture and Implementation. In *Proceedings of the DARPA Active Networks Conference and Exposition (DANCE) 2002*, June 2002. San Francisco, CA, USA.

[73] J. Zander and R. Forchheimer. SOFTNET – An approach to high level packet communication. In *Second ARRL Amateur Radio Computer Networking Conference, San Francisco (USA)*, 1983.

[74] C. F. Tschudin. M0 – a messenger execution environment. Usenet newsgroup comp.sources.unix, Vol 28, Issue 51–62, June 1994.

[75] D. Wetherall, J. Guttag, and D. Tennenhouse. ANTS - A Toolkit for Building and Dynamically Deploying Network Protocols. In *IEEE OPENARCH'98*, 1999.

[76] S. Merugu and S. Bhattacharjee et al. Bowman and CANEs: Implementation of an Active Network. In *37th Annual Allerton Conference*, Sept 1999. Invited paper.

[77] S. Berson, B. Braden, and L. Ricciulli. Introduction to the ABONE, February 2002.

[78] R. Sivakumar, S.W. Hanand, and V. Bharghavan. PROTEAN: A Scalable Architecture for Active Networks. In *OPENARCH'00*, 2000.

[79] D. J. Wetherall and D. L. Tennenhouse. The Active IP Option. In *7th ACM SIGOPS European Workshop*, Sept 1996.

[80] P. Francis. Yoid: Extending the Internet Multicast Architecture. www.aciri.org/yoid/docs/index.html, April 2000.

[81] I. Stoica and R. Morris et al. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *ACM SIGCOMM'01*, pages 149–160, 2001.

[82] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Middleware, LNCS 2218*, pages 329–, 2001.

[83] J. Touch and S. Hotz. Dynamic Internet Overlay Deployment and Management Using the X-Bone. In *ICNP*, pages 59–68, 2000. Osaka Japan.

[84] R. Braynard and D. Kostić et al. Opus: an Overlay Peer Utility Service. In *5th IEEE OPENARCH*, pages 168–178, June 2002. New York.

[85] J.P. Gelas and L. Lefèvre. Performance et dynamicité dans les réseaux : l'approche Tamanoir. In *JDIR 2002*, March 2002. Toulouse, France.

[86] P. Tullmann, M. Hibler, and J. Lepreau. Janos: A Java-oriented OS for Active Networks. *IEEE Journal on Selected Areas of Communication*, 19(3), March 2001.

[87] R. Kilany and A. Serhrouchni. HABA: une architecture à base de composantes distribuées pour le déploiement de services actifs. In *CFIP*, 2002.

[88] S. Martin and G. Leduc. A Dynamic Neighbourhood Discovery Protocol for Active Overlay Networks. In *IWAN'03, LNCS 2982*, pages 151–162, 2003.

[89] D. Wetherall. Active network vision and reality: lessons from a capsule-based system. *Operating Systems Review, ACM*, 33:64–79, Dec 1999.

[90] C. Hawblitzel and C-C. Chang et al. Implementing Multiple Protection Domains in Java. In *USENIX technical conference*, June 1998.

[91] J.T. Moore, J.K. Moore, and S. Nettles. Predictable, Lightweight Management Agents. In *IWAN'02, Zurich, LNCS 2546*, pages 111–119, 2002.

[92] L. Ruf, K. Farkas, H. Hug, and B. Plattner. Network Services on Service Extensible Routers. In *7th Int. Working Conference on Active and Programable Networks (IWAN'05)*, 2005. Sophia Antipolis, France.

[93] M. Bossardt, T. Egawa, H. Otsuki, and B. Plattner. Integrated Service Deployment for Active Networks. In *International Working Conference on Active Networks (IWAN'02), LNCS 2546*, pages 74–86, 2002.

[94] M. Hicks, Kakkar, J. T. Moore, Gunter, and S. Nettles. PLAN: A Packet Language for Active Networks. In *ACM SIGPLAN*, 1998.

[95] J. T. Moore. Safe and Efficient Active Packets. Technical Report MS-CIS-99-24, University of Pennsylvania, October 1999.

[96] J. T. Moore. *Practical Active Packets*. PhD thesis, University of Pennsylvania, 2002.

[97] M. Hicks, J. T. Moore, and S. Nettles. Compiling PLAN to SNAP. In *3rd International Working Conference on Active Networks (IWAN'01)*, September 2001.

[98] A. Kind, R. Plekta, and B. Stiller. The potential of just-in-time compilation in active networks based on network processors. In *5th Workshop on Open Architectures and Network Programming (OPENARCH'02)*, pages 79–90, June 2002.

[99] K. L. Calvert, J. N. Griffioen, and Su Wen. Lightweight network support for scalable end-to-end services. In *ACM SIGCOMM, 2002*, 2002.

[100] S. Wen, J. Griffioen, and K. Calvert. Building Multicast Services from Unicast Forwarding and Ephemeral State. In *IEEE OPENARCH'01*, April 2001. Anchorage, Alaska, USA.

[101] S. Martin and G. Leduc. Interpreted Active Packets for Ephemeral State Processing Routers. In *7th Int. Working Conference on Active and Programmable Networks (IWAN'05)*, 2005.

[102] M. Arlitt and C. Williamson. Web server workload characterization: the search for invariants. In *ACM SIGMETRICS '95*, 1995.

[103] M. Seltzer and J. Gwertzman. The case for geographical push caching. In *Hot Operating System*, 1995.

[104] E. Zegura S. Bhattacharjee, K. Calvert. Self-Organizing Wide-Area Network Caches. In *INFOCOM 1998*, pages 600–608, 1998.

[105] E. Amir, S. Mc Canne, and R. Katz. An Active Service Framework and its Application to Real-time Multimedia Transcoding. In *ACM SIGCOMM'98*, 1998. Vancouver, USA.

[106] S. Bhattacharjee, K. Calvert, and E. Zegura. On Active Networking and Congestion. Technical Report GIT-CC-96/02, Georgia Institute of Technology, ftp://ftp.cc.gatech.edu/pub/coc/tech_reports/1996-/GIT-CC-96-02.ps.Z, 1996.

[107] A. Banchs, W. Effelsberg, C. F. Tschudin, and V. Turau. Multicasting Multimedia Streams with Active Networks. In *IEEE Local Computer Networks Conference LCN'98*, pages 150–, 1998.

[108] E. Gelenbe, R. Lent, and Z. Xu. Design and Performance of Networks with Cognitive Packets. *Elsevier Performance Evaluation*, (46)2-3:155–176, 2001.

[109] E. Gelenbe, R. Lent, and A. Nunez. Self-Aware Networks and QoS. *Proceedings of the IEEE*, 92(9):1478–1489, September 2004.

[110] I. Dedinski and H. De Meer et al. Cross-Layer Peer-to-Peer Traffic Identification and Optimization Based on Active Networking. In *7th International Working Conference on Active and Programmable Networks (IWAN'05)*, 2005.

[111] B. Schwartz, W. Zhou, A. Jackson, W. Strayer, D. Rockwell, and C. Partridge. Smart packets for active networks. In *IEEE OPENARCH'99*, March 1999.

[112] A. Ribeiro Cardoso, A. Serhrouchni, M. Salaün, and J. Celestino Jr. P2P Overlay Network for Service Deployment in Active Networks. 2006. Télécom-Paris, LTCI-UMR 5141 CNRS, Paris, FRANCE.

[113] R. Govindan, C. Alaettinoglu, and D. Estrin. Self-configuring active network monitoring (SCAN), February 1997. White paper.

[114] S. Cheshire and M. Krochmal. DNS-Based Service Discovery. Internet-Draft (work in progress), 2005.

[115] S. Martin and G. Leduc. An Active Platform as Middleware for Services and Communities Discovery. In *International Conference on Computational Science 2005 LNCS 3516 (part 3)*, pages 237–245, 2005.

[116] C. Partridge, T. Mendez, and W. Millikenn. RFC-1546 - Host Anycasting Service, November 1993.

[117] T. Dübendorfer, M. Bossardt, and B. Plattner. Adaptive Distributed Traffic Control Service for DDoS Attack Mitigation. In *SSN 2005*, April 2005. Denver, USA.

[118] L. Xie, P. Smith, J. Sterbenz, and D. Hutchison. Building Resilient Networks using Programmable Networking Technologies. In *7th International Working Conference on Active and Programmable Networks (IWAN'05)*, 2005.

[119] J. Lockwood and J. Moscola et al. Application of Hardware Accelerated Extensible Network Nodes for Internet Worm and Virus Protection. In *International Working Conference on Active Networks (IWAN'03)*, December 2003.

[120] K. Hwang and Y.-K. Kwok et al. GridSec: Trusted Grid Computing with Security Binding and Self-Defense against Network Worms and DDoS Attacks. In *International Workshop on Grid Computing Security and Resource Management (GSRM'05), in conjunction with the ICCS-2005*, pages 187–195, 2005.

[121] L. Yamamoto and C. F. Tschudin. Experiments on the automatic evolution of protocols using genetic programming. In *Proceedings of the 2nd Workshop on Autonomic Communication (WAC 2005)*, Athens, Greece, October 2005.

[122] Haggle Project - http://www.haggleproject.org/.

[123] Bionets Project - http://www.bionets.eu/.

[124] Cascadas Project - http://www.cascadas-project.org/.

[125] Ambient Networks - http://www.ambient-networks.org/.

[126] BISON - http://www.cs.unibo.it/bison/.

[127] Internet2 - http://www.internet2.edu/.

[128] GENI - http://www.geni.net/.

[129] T. Anderson, L. Peterson, S. Shenker, and J. Turner (Editors). Report of NSF Workshop on Overcoming Barriers to Disruptive Innovation in Networking, January 2005. Available at http://www.geni.net/documents.php.