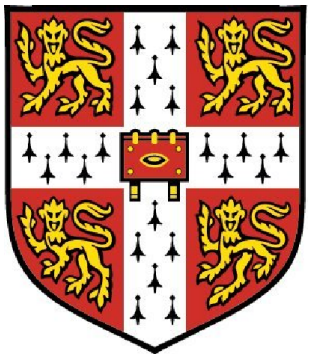
Scalable Software Defined Networking



Charalampos Rotsos

Churchill College

University of Cambridge

A thesis submitted for the degree of

Doctor of Philosophy

June 2014

Scalable Software Defined Networking

Charalampos Rotsos

Computer network technologies have become the catalyst for the development of the current digital revolution. Nonetheless, their increased adoption has highlighted novel challenges in their functionality. The increasing popularity of the network abstraction has unveiled a series of scalability limitations in the design of the predominant protocols. In addition, the increase in the number of network applications has further complicated the definition of network performance, i.e. management flexibility, resource control, connectivity.

The thesis of this dissertation contends that the perfomance scalability limitation of data plane protocols can be alleviated by redesigning the control plane architecture of networks. We argue for the development of specialised control designs tailored to the requirements and the opportunities of the deployed environment, taking advantage of the uprecedented capabilities provided by the SDN paradigm for programmable control.

Towards this goal, this dissertation explores three specific cases of scaling networks for improved perfromance by evolving network control. Firstly, we present two generic tools that enable characterisation of the scalability and performance of programmable network control. Using these tools, we characterise the scalability of the control plane performance for a series of production OpenFlow-enabled forwarding devices and the performance of hierarchical control schemes in a datacenter environment. Secondly, we present a control plane design that scales resource and access management for the home network environment. Finally, we present a control plane design providing Internet-wide naming and connectivity.

Declaration

This thesis:

- is my own work and contains nothing which is the outcome of work done in collaboration with others, except where specific in the text;
- is not substantially the same as any that I have submitted for a degree or diploma or other qualification at any other university; and
- does not exceed the prescribed limit of 60,000 words.

Charalampos Rotsos
June 2014

Acknowledgements

I would like to thank my supervisor Andrew W. Moore for his guidance, support and significant patience during my PhD. In addition, I wish to express my gratitude for the excellent internship opportunities provided by Richard Mortier, Steve Uhlig, and Peter Neuman in the University of Notingham, the TU-Berlin, and the SRI. I wish also to thank Nadi Sarrar and Rob Sherwood for their help in OFLOPS, Dimos Pediadiatakis for his fruitful input to SDNSIM and its descendents, and Narseo Vallina-Rodriguez, Anil Madhavapeddy, Sebastian Probst Eide, Jon Crowcroft, Heidi Howards, and Andrius Aucinas for their collaboration in the early stages of design of Signpost. Finally, I would like to thank Simon Moore and Robert N. M. Watson for the opportunities they provided me to work in the $(MRC)^2$ project during my write up year.

I would like to express my gratitude to the following people for their academic and moral support: Christos Efstratiou, George Parisis, Georgina Kalogeridou, Jury Audzevich, Gianni Antichi, Salvatore Scelato, Ilias Leontiadis, Liam McNamara, Tasos Noulas, Ilias Marinos, Maria Bakali.

I would also like to thank Elisavet Christou for her support and patience during a difficult period of our life. Last but not least, I wish to thank my family, Ioannis Rotsos and Olympia Lianoudaki-Rotsou, for their continuous support and encouragement throughtout my life. This work is dedicated to Elisavet and my parents.

Contents

List of Figures

List of Tables

List of Publications

As part of my PhD work I have published the following work:

- **C. Rotsos**, J. Van Gael, A. W. Moore, and Z. Ghahramani. *Probabilistic graphical models for semi-supervised traffic classification*. In Proceedings of the 6th International Wireless Communications and Mobile Computing Conference (IWCMC), 2010.

- R. Mortier, B. Bedwell, K. Glover, T. Lodge, T. Rodden, **C. Rotsos**, A. W. Moore, A. Koliousis, and J. Sventek. *DEMO: Supporting novel home network management interfaces with OpenFlow and NOX*. In the Proceedings of the ACM SIGCOMM 2011 conference.

- R. Mortier, T. Rodden, T. Lodge, D. McAuley, **C. Rotsos**, A. W. Moore, A. Koliousis and J. Sventek, *Control and understanding: Owning your home network*. In the Proceedings of the 4th International Conference on Communication Systems and Networks (COMSNETS), 2012.

- **C. Rotsos**, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore. *OFLOPS: an open framework for OpenFlow switch evaluation*. In Proceedings of the 13th international conference on Passive and Active Measurement (PAM), 2012.

- A. Chaudhry, A. Madhavapeddy, **C. Rotsos**, R. Mortier, A. Aucinas, J. Crowcroft, S. P. Eide, S. Hand, A. W. Moore, and N. Vallina-Rodriguez. *DEMO: Signposts: end-to-end networking in a world of middleboxes*. In the Proceedings of the ACM SIGCOMM 2012 conference.

- **C. Rotsos**, R. Mortier, A. Madhavapeddy, B. Singh, and A. W. Moore. *Cost, performance & flexibility in OpenFlow: Pick three*. In the Proceedings of the 2012 IEEE International Conference on Communications (ICC), 2012.

- A. Madhavapeddy, R. Mortier, **C. Rotsos**, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft. *Unikernels: library operating systems for the cloud*. SIGPLAN Not. 48, 4 (March 2013).

- **C. Rotsos**, H. Howard, D. Sheets, R. Mortier, A. Madhavapeddy, Amir Chaudhry, and J. Crowcroft. *Lost in the edge: Finding your way with DNSSEC signposts*. In the Proceedings of the 3rd USENIX Workshop on Free and Open Communications on the Internet, 2013.

- A. Sathiaseelan, **C. Rotsos**, C. S. Sriram, D. Trossen, P. Papadimitriou, and J. Crowcroft. *Virtual Public Networks*. In the Proceedings of the 2nd European Workshop on Software Defined Networks (EWSDN), 2013.

Chapter 1

Introduction

Computer networks have become the predominant communication medium of the digital era. Nonetheless, the widening adoption of network technologies has highlighted several scalability problems in network functionality. These problems primarily derive from the design of network protocols, and architectures. Addressing these limitations through protocol redesign is a challenging process, which requires significant time and effort for deployment. Network systems require novel backwards-compatible mechanisms that provide scalable performance.

This dissertation is concerned with performance scalability problems in modern networks. The thesis of this dissertation is that control plane redesign can mitigate network bottlenecks, introduced by protocol design, while ensuring backwards compatibility and high performance. We argue that existing control plane mechanisms remain inflexible and that their "single abstraction fits all" approach fails to fulfil the unique requirements of a set of novel deployment environments, e.g. datacenters and home networks. We advocate that networks require control redesign, *tailored* to the requirement and opportunities of the environment. The recent rebirth of SDN in network devices provides a sufficient and readily-available enabler for the proposed approach.

For the remainder of this chapter, we discuss the motivations of our work (§ 1.1), followed by the explicit definition of our contributions (§ 1.2) and the outline of this thesis (§ 1.3).

1.1 Motivation

This section presents the motivations for our thesis. Specifically, motivated by the radical evolution of current network technologies(§ 1.1.1), we discuss the resulting network performance complexity (§ 1.1.2) and exemplify some of the limitation incurred by the design of Internet protocols (§ 1.1.3).

1.1.1 Computer Network Evolution

One of the key mechanisms that formed the objective conditions for the digital revolution of our era was the paradigm of packet-switched computer networking. Computer networks provide a generic data-exchange mechanism between 2 computers over a physical medium. The most successful attempt to define and implement a heterogeneous computer network of significant size was *ARPANET* [Beranek, 1981]. ARPANET improved the resilience and performance of available circuit-switched networks, and its simple design allowed easy integration with existing computing systems. It was adopted by a small number of US education and research institutes and allowed, for the first time in computing history, to interconnect multiple mainframes using a packet-switched network. The ARPANET community standardized a small set of data-exchange protocols, namely: e-mail [Bhushan et al., 1973], FTP [Bhushan, 1972] and voice [Cohen, 1977]. ARPANET was later replaced by the NSFNET [Mills and Braun, 1987], which eventually evolved in today's Internet. As part of this transition, the research community developed the standards for the Internet protocol suite [Braden, 1989; Clark, 1988; Hornig, 1984; Postel, 1981a,b], the lower and middle layers of the network stack.

Computer networks have gained a significant position in our society since the ARPANET years, co-evolving with the digital era. Their communication abstraction replaces a number of traditional communication systems, and provides support for a wide range of applications and deployment environments (e.g. telephone networks). As a result, in 2011 a third of the global population is Internet-connected through a wide range of network technologies [ITU, 2011]. The average user is Internet-connected in parallel through multiple devices (e.g. laptop, smartphone, home entertainment system) and a large portion of daily activities rely on network technologies

Application	Throughput (Mbps)	Latency (sec)	Jitter (sec)	Flow Number (median)
Web [Butkiewicz et al., 2011; Research and Akamai, 2006]	-	4	-	10
Video [Finamore et al., 2011]	0.3 - 5	-	-	1
Peer-to-peer [Pouwelse et al., 2004; Rasti and Rejaie, 2007]	-	-	-	30
VoIP	0.1 - 1.5	0.5	0.1	1
Gaming [Armitage et al., 2006]	0.1	0.1	0.05	1

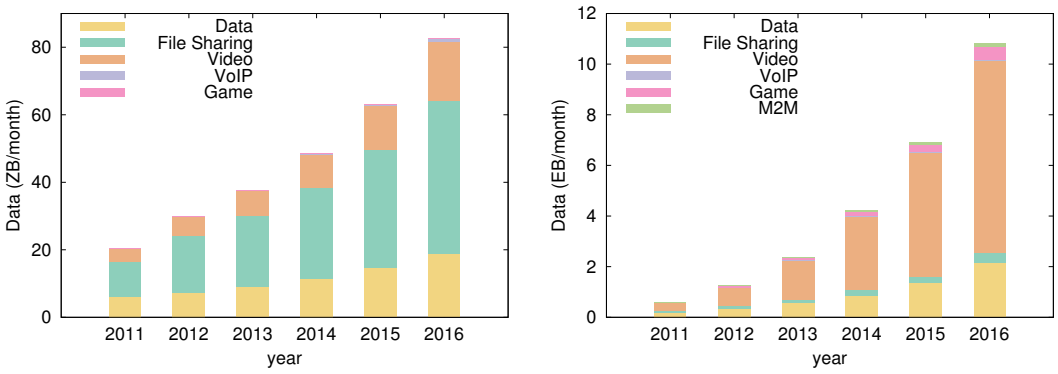
Table 1.1: Network performance requirements for popular traffic classes.

(e.g. e-banking, e-gov, online social networking). The importance of computer networks is further reflected in its importance to the global economy. Internet resources are estimated to generate 3.4% of global GDP [du Rausas et al., 2011], while broadband availability increase in developing economies exhibits a positive correlation with GDP growth (10% increase in the number of households connected to broadband services improves GDP by approximately 1% [Katz, 2011]). Nonetheless, the growth in network adoption introduces novel performance requirements for network technologies and protocols.

1.1.2 The Lernaean Hydra of Network Performance

Traditional computer network theory textbooks (such as [Peterson and Davie, 2011]) define network performance using simple objective metrics, like latency, bandwidth, jitter and packet loss. Nevertheless, the increased adoption of network technologies creates an equal augmentation in network use-cases, and the definition of performance must preclude the end-user and application perspective. Consequently, the definition of network performance becomes complex and is domain and application-specific, considering additional subjective aspects, i.e. usability. We elaborate on the complexity and multidimensionality of performance by discussing two example aspects: application resource requirements and network heterogeneity.

The availability of the network abstraction in multiple devices and contexts motivates the development of a wide range of network applications with global audiences



(a) Fixed Internet

(b) Mobile Internet

Figure 1.1: Cisco Visual Network Index report on global traffic trends for fixed (Figure 1.1(b)) and mobile Internet (Figure 1.1(a)).

and diverse performance requirements. As an example, Table 1.1 presents the network requirements in terms of bandwidth, latency, jitter and number of concurrent flows for web, video streaming, peer-to-peer, VoIP and gaming application classes. Based on the table, it is evident that current network applications have contradicting requirements. For example, peer-to-peer applications do not have any performance requirements but try aggressively to maximize network throughput, affecting applications with strict latency requirements, like VoIP. As a result, applications face difficulties in sharing network resources during times of congestion. Network operators overcome the sharing problem by over-provisioning bandwidth resources, but physical limitations reduce the effectiveness of such approaches.

Application performance exhibits similar complexity at a macroscopic level. Figure 1.1 presents estimated traffic volumes for six popular network application classes between the years 2011 and 2016 for mobile and fixed Internet, using data from the Cisco visualization index [CVNI, 2012a,b]. Aggregate network traffic volumes exhibit an exponential augmentation and is expected to increase by an order of magnitude for mobile Internet, and by four times for fixed Internet between the years 2011 and 2016. Traffic volume increase is primarily driven by video delivery applications, which are increasing in popularity over file-sharing applications. The popularity churn reflects the availability of predominant file-sharing video content by high-availability CDN services like YouTube. Even so, the two classes exhibit different characteristics with

respect to bandwidth requirements, and thus network providers are obliged to reflect these trends in their network architecture and configuration. Furthermore, application traffic volumes commonly exhibit long-term popularity fluctuation, but a minimum traffic volume will always exist in the network and shape ISP policies and architectures.

A significant evolution also exists in the lower layers of the network. Ethernet has been the predominant Internet link layer protocol since the 80's, mainly due to its low cost and complexity. The network community has defined standards to implement the Ethernet abstraction over a wide range of medium types, like copper, fibre, off-licence radio frequencies, satellite links and GSM networks, encapsulating significant heterogeneity. This approach hides a number of medium properties (e.g. link layer ARQ, link rate), which reflect significant performance characteristics of the link layer. At the same time, applications, network layer protocols and users remain unaware of these factors and require homogeneous performance semantics.

1.1.3 Performance Limitations

Current network protocols have a number of design limitations, primarily due to the inability to predict the radical adoption of the technology during their development. The early adopters of the NSFNET, the incubator of Internet protocol standards, were universities and research facilities, and the resulting system aimed to support asynchronous communication, low data-rates, open service connectivity and best-effort guarantees. These initial assumptions have been radically revised in the Internet over the years but the protocol designs have not managed to keep abreast. The mismatch between protocol capabilities and current requirements grows in proportion to the increase of link speeds.

In parallel, the evolution of network technologies, and especially of their control plane, has highlighted that the adopters dictate abstraction simplicity as a primary design goal. The Internet protocol definition process followed a Darwinian selection approach between available solutions during the early days of NSFNET. Along with the TCP/IP protocol suite, OSI and ITU proposed equivalent protocol stacks [ISO/IEC, 1997, 2001], while the ATM Forum also defined similar higher-layer protocols [Siu and Jain, 1995]. These protocol design efforts defined detailed abstractions which

addressed some current functional limitations. For example, ATM provided strong resource guarantees, motivated by the fixed cell size and circuit-based forwarding approach. Nonetheless, their high complexity faces performance scalability problems with respect to available computational resources, and the protocols became obsolete in favour of the TCP/IP protocol suite. For the rest of the section we will discuss in detail some examples of network bottlenecks, that have arisen as a consequence of the design of current network protocols.

Management Scalability Policy and configuration interfaces in current networks introduce a significant performance bottleneck. Mahajan et al. [2002] pinpoint a significant portion of global BGP routing errors to protocol misconfiguration, prompted to a great extent by the counter-intuitive user interface. Kim et al. [2011a] study the policy evolution of two medium-sized campus networks and highlight the limitations of existing interfaces. Device configuration files exhibit a continual size increase, containing frequently stale rules that are pruned only during policy conflicts or major policy revisions. Interfaces remain low-level and inexpressive. As a result, network policy updates translate into multiple device reconfigurations, while forwarding and security policies are merged in the device interface. The management bottleneck is equally important in environments with non-expert administrators, like the home network. Grinter et al. [2005] conclude that the human understanding of network abstraction is limited and the complexity of control abstraction leads to under-optimized network configurations, especially in terms of security and performance.

Currently, network management relies extensively on the network administrator, who is responsible for manually transforming the network policy into specialised configurations for each network device and network layer. For example, the policy must be translated both into VLAN tagging configurations in the data-link layer, and routing and firewall configurations for the network layer. Section 2.1 provides an extensive presentation of current network control mechanisms and available control abstractions. The rapid development of network technologies has limited effect on device interfaces, which remain low-level and counter-intuitive to administrators. As a result, network management still remains a process requiring and relying on the experience of the network administrator. Effectively, network management interfaces introduce a bottleneck on the "mechanism" translating the network policy into distributed device

configurations: the *n*etwork administrator.

Network management requires a technological "revolution" similar to the introduction of high-level programming languages in the 70's. In order to scale network management performance, we require new, user-friendly control abstractions, which can automate the process of policy translation, unify the control domain across the network and specialize the control plane functionality to the deployment environment.

Resource Allocation As discussed in Section 1.1.2, modern network functionality requires dynamic, fine grain and rapid resource control in order to fulfil the diverse application requirements. Network engineers commonly address this problem through resource over-provision [Teitelbaum and Shalunov, 2002], which, however, exhibits reduced effectiveness at high link-rates. High access speeds increase application responsiveness and similarly shape end-user expectations. Although, during high link utilisation incidents, network responsiveness and user experience degrade severely, in the vicinity of queueing delays and packet losses. A number of mechanisms have been proposed to improve resource control, in various layers of the network stack (e.g. ECN [Kuzmanovic et al., 2009], DiffServ [Blake et al., 1998], RSVP over MPLS-TE [Awduche et al., 2001]). These mechanisms however, for different reasons, fail to provide a generic solution to the problem.

The ideal mechanism for effective resource control requires a priori knowledge of resource requirements from end-hosts in order to define optimal resource allocations. Currently, networks follow a distributed approximation using a feedback mechanism between two systems: the end-to-end congestion control; and the network forwarding policy. End-to-end congestion control infers the available bandwidth of the network path using flow properties, like packet loss and RTT variance, and adjusts the traffic rate accordingly. The network forwarding policy defines, statically or dynamically, network paths between hosts, as well as prioritisation and buffer size for specific network flows. Forwarding policy decisions can modify flow properties, affect end-to-end congestion control and enforce resource control policies effectively. Nonetheless, because the two mechanisms are weakly integrated, policy enforcement is effective in long timescales. Resource management in current networks requires smarter control planes, which augment the inputs of the decision algorithm and provide faster responses.

Connectivity scalability A popular approach to address network scalability problems uses middleboxes; transparent network devices which redefine the behaviour of data plane protocols, like NAT boxes, WAN optimizers and firewalls. Middleboxes have, to a great extend, redefined overall protocol functionality, as they commonly introduce new assumptions into the protocol functionality and violate the "Robustness principle" of networks[1]. In addition, their functionality and behaviour varies significantly between vendors, and remains inflexible and isolated from the control plane of the operating network. Honda et al. [2011] present an Internet-wide robustness measurement study and report that obscure but semantically correct protocol behaviours are unsupported by a number of edge networks, primarily due to the functionality of such devices. Similarly, Hatonen et al. [2010] study the operational semantics of a series of home network routers and highlight significant differences between commercial NAT implementations.

A significant side-effect of middlebox functionality is the redefinition of elementary network properties, like the bi-directionality of network connectivity. For example, NAT boxes improve address scalability but reduce host reachability. All hosts behind a NAT are able to access any Internet service using a single IP address, but require network reconfiguration to expose an Internet-wide service. A series of protocols has been proposed to enable end-hosts middlebox control [Carpenter and Brim, 2002], but their generality is limited and context-specific.

1.2 Contributions

This dissertation contends that existing network architectures cause a significant performance bottleneck in network functionality and provide limited support to the constantly evolving network requirements of end-users. Modern networks require flexible and specialized control approaches that address complex functional requirements, and unify network control across all devices of the network. Our study employs the SDN control approach and the OpenFlow protocol extensively. Nonetheless, the applicabil-

[1] "In general, an implementation must be conservative in its sending behavior, and liberal in its receiving behavior. That is, it must be careful to send well-formed datagrams, but must accept any datagram that it can interpret (e.g., not object to technical errors where the meaning is still clear)." [Postel, 1981a]

ity of our contributions are not limited to the OpenFlow protocol and can easily adapt to alternative programmable control frameworks. In detail, we claim the following specific contributions:

- **Control plane scalability**: We provide an in-depth study of SDN control scalability. In this context, we present two measurement and evaluation platforms for SDN control: OFLOPS; and SDNSIM. OFLOPS is a high precision switch evaluation platform, providing a range of testing modules for elementary OpenFlow interactions. SDNSIM is a network experimentation platform, specialised for OpenFlow control architectures, which allows a user to easily describe and either simulate or emulate an experimental setup. These two platforms establish a generic toolbox, providing a flexible evaluation of SDN control applications. Using these tools, we provide an in-depth analysis of control plane performance and limitations for a wide range of off-the-self OpenFlow implementations. Additionally, using as an input the OFLOPS performance models, we explore the impact of a hierarchical reactive control scheme on the performance of a small-scale datacenter.

- **Network management scalability**: We present a control application which addresses the problem of network management and resource control. We consider the multiple and varied domain of the home network. Motivated by recent ethnographic and measurement studies for home networks, we identify user requirements for accurate network state information and intuitive control primitives. We address these requirements through the home router by redesigning the control plane functionality. While maintaining full compatibility with existing network applications, our design provides a user intuitive control abstraction, which incorporates the user's social input into the forwarding decision and improves both access and QoS control.

- **Naming and Connectivity scalability**: We revisit the problem of connectivity and naming scalability in the Internet. Our exploration is motivated by the requirement of end-users to interconnect devices across the Internet in an ad hoc manner, thus forming Personal Clouds. Currently, due to connectivity limitations of the Internet, this requirement is fulfilled through data offloading to

third party cloud services, raising concerns regarding performance, efficiency and security. We propose Signpost, a distributed control plane architecture for end-hosts, which provides inter-device persistent naming and connectivity and controllable performance and security of the established network. Simple augmentation of end-host forwarding logic, allows Signpost to integrate seamlessly with a wide range of existing resource sharing applications.

1.3 Outline

The rest of this dissertation is organised as follows. Chapter 2 provides background information related to our thesis. Motivated by the architecture of current network devices and the physical and design limitations of network control, we revisit control plane mechanisms and related research efforts into control plane scalability.

Chapter 3 studies the control scalability of the SDN paradigm. We present two control plane benchmarking platforms, *OFLOPS* and *SDNSIM*, designed to provide high precision evaluation of the scalability of OpenFlow devices and architectures, respectively. Using these tools, we conduct a measurement study to characterise the performance of SDN-enabled devices with respect to baseline OpenFlow operations and the impact of distributed control architectures in data plane performance. Chapter 4 studies the problem of control plane management performance, focusing on the home network setting. Using existing user studies, we draw out the user network management requirements and redesign accordingly the control plane architecture of the home router. Our architecture provides a simpler, user-friendly control abstraction which fulfils the user requirements for control and information from their network. In addition, we provide an extensive evaluation of the architecture, providing evidence on scalability and backwards compatibility. Chapter 5 investigates the application of the SDN paradigm in naming and connectivity scalability to create a federated network between a user's devices. We present Signpost, a novel user-centric control plane architecture, providing Internet-wide device connectivity with user-controlled performance and security. We also present the architecture of the framework, its integration with existing connectivity-enabling mechanisms and we evaluate the performance and applicability of the framework. Finally, Chapter 6 draws conclusions from our research and discuss areas for further study.

Chapter 2

Background

This chapter provides an extensive discussion on network control in packet-switched networks. We motivate the discussion on network control, describing the architecture of network devices and their inherent physical limitations (Section 2.1) and elaborate on existing network control mechanisms for production networks (Section 2.2). Furthermore, we present three significant efforts in flexible network control, namely Active Network, Devolved Control of ATM Networks (DCAN) and Software Defined Networking (SDN), proposed by the research community to address limitations in network control schemes (Section 2.2). Finally, we focus on the SDN paradigm and present some of its applications in a series of current network problems (Section 2.4).

2.1 Forwarding Devices and Control

This section focuses on Ethernet networks and provides a high-level design overview of a common forwarding device, the *switch*. Using this model we highlight the physical limits of network performance in the control and data plane and motivate our discussion on control effectiveness.

Ethernet switches multiplex Ethernet broadcast domains and provide collision-free connectivity between network segments. Network vendors provide a wide range of switch types with variable functional capabilities (e.g. VLAN support, multilayer switches). For example, an unmanaged switch provides elementary non-configurable functionality with low 1 GbE port density, while a distribution switch provides high

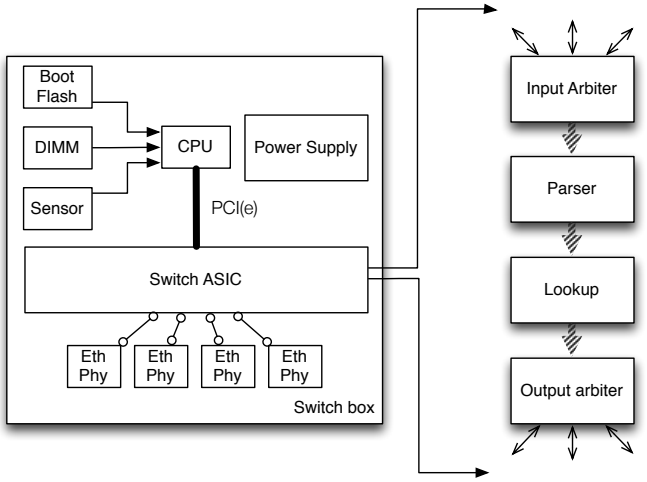


Figure 2.1: Generic design model of hardware switches.

10 or 40 GbE port density and a range of traffic management applications. We use the Top-of-Rack (ToR) switch type to elaborate on the architecture of modern hardware switches. ToR switches are used in datacenter and enterprise networks to multiplex traffic between the edge and distribution network layers. Forwarding functionality is implemented using Application-Specific Integrated Circuit (ASIC) silicons, providing multi-GbE line-rate non-blocking traffic forwarding. Figure 2.1 presents a model for ToR switch architectures which consists of the following components:

- *Co-processor/Management CPU*: Switch devices use a programmable CPU to fulfil control plane computation requirements. The CPU hosts a minimal operating system, responsible for translating the device configuration into appropriate ASIC manipulations at run-time. Vendors usually employ low-power CPUs, like SoC PowerPC and ARM processors. The CPU is equipped with runtime (Boot Flash and RAM) and persistent (e.g. SD cards) memory modules. In general, switch CPUs provide sufficient computation resources to run the switch control-plane functionality, but they cannot accommodate intensive processing tasks. The switch OS usually provides elementary remote control services, like telnet and SSH, command line/web interfaces and SNMP access to the switch state.

- *Switch ASIC*: The switch ASIC [Broadcom, 2013; HP, 2012; Intel, 2014] implements in hardware the data plane functionality. The capabilities of an ASIC

are variable, depending on the vendor and the cost, and define the data plane performance limits of the device. ASICs have an expensive development cycle, consisting of long design and testing periods, and require significant human labour and material resources requirements. As a result, ASIC innovation has a high latency before reaching production, while the design details remain undisclosed.

The ASIC packet processing pipeline commonly consists of four stages. The *input arbiter* stage multiplexes and synchronizes packets from the Ethernet ports to the main processing pipeline of the silicon. The arbiter ensures non pre-emptive packet processing, by bridging the mismatch between the silicon processing clock rate and the link rate. The pipeline also contains a *protocol parsing* stage and a *memory lookup* stage. The protocol parser extracts significant packet fields from network packets, used by the memory lookup module to define packet processing and forwarding. The lookup stage uses a memory module, integrated or external to the ASIC, which contains the forwarding policy. Memory modules exhibit trade-offs between cost and access speed and memory cost and memory management complexity[1]. Finally, the *output arbiter* module is responsible for applying modifications to the packet and forwarding it to the appropriate output queue. The processing pipeline may contain additional modules which extend functionality, providing access control list (ACL), virtual network queues, and flow statistics monitoring.

- *ASIC-CPU interconnection*: The management CPU is connected to the ASIC over a PCI or PCI-express channel. The interconnection provides an elementary bi-directional channel allowing the CPU to program the ASIC through its register API, and the ASIC to propagate exceptional packets to the CPU. The channel provides sufficient bandwidth for basic control plane communication, but it is not designed to handle high rate information. For example, current Broadcom Trident chipsets use a PCIe 4x or 8x bus, achieving 2 to 4 Gbps capacity.

[1] A 32-bit TCAM lookup pcore running on a Virtex-5 FPGA [Xilinx Inc., 2009] with 550 MHz clock rate requires 1 clock cycle for a lookup, translating into a 2 ns delay. A DDR3-2400 SDRAM memory module operating at 1200 MHz and requiring 9 clock cycles per data access has a 9 ns lookup latency, while a PC133 SDRAM operating at 133 MHz and requiring 3 clock cycles per data access has a 20 ns lookup latency.

- *Memory*: Apart from in-ASIC memory modules, switches also use multiple memory types to support the run-time requirements of the firmware. A switch is usually equipped with CPU Boot Flash, to boot the OS/firmware, DIMM RAM and multiple memory slots, for control plane logging and configuration persistence.

- *Ethernet ports*: The receipt and transmission of Ethernet packets is implemented by a separate hardware module, implementing the physical and MAC layers of the protocol. The port module connects with the ASIC through a Media Independent Interface (MII) and contains small packet buffers to reduce packet loss.

In terms of data plane performance, the forwarding capabilities of an ASIC are upper-bound by the transistor density and size, as in general purpose CPU silicones. A ToR switch uses a single ASIC, supporting up to 100 Gbps of processing capacity. In terms of control plane performance, the primary limitations are the capacity of the communication bus between the coprocessor and the ASIC, the design of the interface between the two entities, and the processing capabilities of the switch CPU. For a ToR switch, the vendors usually provide a PCIe communication bus, supporting capacities on the order of a few Gbps, while CPUs have limited processing capabilities (Section 3.3 provides details on some off-the-self ToR switches).

ToR switches are simple switch devices and provide low to medium forwarding capacity. Vendors provide network devices that support higher link capacities and enhanced functionalities that have more complex architectures. Multi-chassis switches and routers, used in the core of large networks, cannot fit their processing pipeline requirements in a single ASIC. Packet processing is distributed between multiple silicons, interconnected using complex CLOS crossbar fabrics with non-blocking Terabit backplane capacities [Juniper, 2012]. Network control in such devices exhibits similar high complexity, and uses distributed forwarding tables and cache coherent protocols to ensure state consistency [Cisco, 2005]. In such devices, ASIC control is complex, inflexible and slow.

2.2 Forwarding Control in Production Networks

Current network architectures and technologies set dynamic control as a fundamental design goal. Existing standardized approaches provide *distributed*, *autonomous* and *resilient* control, influenced extensively by the end-to-end Internet principle. The administrator is responsible for defining the local policy of a device, and at run-time distributed protocols reconstruct the global network state. Using the global network state, the forwarding logic determines an optimal forwarding policy which optimizes specific aspects of network performance. For the rest of this section we present the control plane functionality in the data-link and network layers of the network stack.

2.2.1 Data-Link Layer Control

Data-link layer control protocols provide loop-detection, VLAN and QoS configuration automation. Loop detection is used to avoid packet loops in networks with redundant connectivity, as Ethernet specification doesn't support packet timeout functionality. The Spanning Tree Protocol (STP) was a first attempt to address this problem, standardised by IEEE in 802.1D-1998 [IEEE, 1998]. The protocol implements the distributed spanning tree construction algorithm, presented in [Perlman, 1985]. The algorithm uses broadcast messages to discover a spanning tree over the network graph, with respect to a common root switch. For each port, the switch maintains a state machine, which disables packet forwarding when the respective link is not part of the spanning tree. Because the initial definition of the protocol faced significant convergence delay after a network change, IEEE introduced the Rapid Spanning Tree Protocol (RSTP), an evolved version of STP [IEEE, 2004]. In addition, IEEE Multiple Spanning Tree Protocol (MSTP) [IEEE, 2005] developed a modified version of the protocol, optimized for VLAN-based networks. MSTP constructs a spanning tree for each VLAN, which effectively reduces unnecessary port blocking. Cisco has developed a series of proprietary protocols to address the problem in a similar manner [Cisco, 2012c,d]. Finally, IEEE recently defined an STP protocol to detect and use redundant paths in the IEEE 802.1aq standard [IEEE, 2006].

In terms of configuration automation, network vendors and standardization bodies have developed a number of protocols to disseminate device status between neigh-

bouring nodes. This functionality differs significantly between vendors. In this class of protocols we consider Link Layer Discovery Protocol (LLDP) [IEEE, 2009], Cisco Discovery Protocol (CDP) [Cisco, 2012a], Extreme Discovery Protocol (EDP) and Nortel Discovery Protocol (NDP). Finally Cisco has developed the VLAN Trunking Protocol (VTP), a protocol which reduces the VLAN configuration burden in inter-switch trunk links.

In the class of data-link layer protocols, we also consider the MPLS protocol [Rosen et al., 2001]. MPLS is a circuit-based technology and uses labels to forward packets, effectively reducing forwarding table sizes. MPLS circuit creation is automated through the Label Distribution Protocol (LDP) [Andersson et al., 2007], which uses an underlying routing protocol to compute and setup labels across the network. The IETF defines an RSVP resource allocation mechanism over MPLS [Awduche et al., 2001] and *Autobandwidth* [Osborne and Simha, 2002], an automatic mechanism to enforce such resource allocation. Because MPLS does not support resource policing, autobandwidth monitors the bandwidth requirements for each circuit and reconfigures circuits in order to fulfil measured requirements. Pathak et al. [2011] analyse an autobandwidth deployment on the MSN network and highlight significant latency effects as a result of the autobandwidth functionality.

2.2.2 Network Layer Control

Control in the network layer is responsible for collectively constructing an optimal forwarding table between the routers of a network. Routing protocols are generally categorized into three classes: *Link State*; *Distance Vector*; and *Path Vector*. Link state protocols theory build on-top of the Djikstra algorithm [Dijkstra, 1959]. Routers disseminate their local forwarding configuration to the rest of the network. Using this global state exchange, each router is able to construct the global connection graph. Each router can use the graph to calculate the minimum spanning tree of the network, using Djikstra's algorithm. Currently there is a plethora of Link State protocol specification in the network community. IETF has developed the OSPF protocol [Moy, 1998], an IPv4 specific routing protocol, while the Open Systems Interconnection (OSI) organisation has defined the IS-IS protocol [Oran, 1990], a network layer agnostic routing protocol. Link State routing protocols provide high flexibility in defining the optimi-

sation function of the routing system. Each router has a view over the complete graph of the network, and routers can propagate multiple link performance indexes (e.g. link load, link speed). Nonetheless, the optimization function must be homogeneous across the network, in order to avoid routing loops.

Distance Vector protocols follow a different routing approach, based on Belman-Ford [Bellman, 1956]. The routing table is constructed using information from adjacent routers. Network changes are slowly propagated across the network, through point-to-point information exchanges, until all routers converge. IETF developed the RIP protocol [Malkin, 1998] to implement Distance Vector routing, while Cisco has developed the proprietary IGMP protocol [Hedrick, 1991]. Distance Vector protocols are less extensible in comparison to Link State protocols, but exhibit lower computational and memory requirements.

Finally, Path Vector protocols evolve Distance Vector protocols to support inter-domain routing. Path Vector routing discloses minimal information in terms of the forwarding policy of an Autonomous System (AS), advertising only supported network paths, As a result, Path Vector protocols do not define a routing algorithm. The BGP protocol [Rekhter, 1991] is currently the predominant Path Vector routing implementation. As Internet increases in size, BGP deployment has experienced significant scalability problems and motivated some of the network control frameworks presented in Section 2.2. A significant scalability problem in BGP is the impact of path updates in the functionality of routers in large ASes. Path updates are handed by the iBGP protocol using a full-mess dissemination approach, which incurs significant router load and memory usage. A widely used approach in addressing this problem uses Route Reflectors (RR) [Bates et al., 2006] hosts, which form a hierarchical update dissemination mechanism and reduce forwarding information on each router. A further improvement on iBGP scalability was developed by Caesar et al. [2005] as part of the Routing Control Platform (RCP), proposing control centralisation similar to the SDN paradigm. RCP basically proposed the centralisation of the BGP routing table calculation within an AS.

Due to their distributed nature, routing protocols provide long-term routing resilience, while their mathematical foundations are provably correct. Nonetheless, the control abstraction of these protocols is not a good fit for an administrator to exercise dynamic control over the network. For example, the ability of a network manager to

estimate the impact of a significant forwarding policy update is reduced as the network size increases. In 2008, the global Internet was severely affected by a BGP misconfiguration in the Pakistani national ISP in an effort to control traffic from the YouTube service [Brown, 2008]. In addition, routing inconsistencies during routing changes can significantly impact network performance. Watson et al. [2003] evaluate that OSPF functionality in a regional ISP exhibits high routing churn, even during periods with low control plane activity, while the latency in convergence after a network change is, on average, on the order of seconds. Similar results have been observed in BGP. Kushman et al. [2007] highlight a strong correlation of BGP instability and VoIP performance. Modern high speed networks require higher flexibility and responsiveness in the control abstraction.

2.3 Programmable Network Control

The limitations of current standardized network control frameworks have motivated the research community to reconsider their design. The rest of this section presents three significant efforts, namely Active Network, Devolved Control of ATM Networks (DCAN) and Software Defined Networking (SDN).

2.3.1 Active Network

The network research community has highlighted the "inevolvability" of network functionality, often termed *protocol ossification*, since the early 90's. O'Malley and Peterson [1992] challenged the generality of the OSI network model and suggested synthesizable protocol parsers, which can incorporate variable numbers of layers in forwarding devices. Motivated by these observations, DARPA funded the *Active Network* project [DARPA, 1997] to develop next generation network devices supporting seamless network upgradability.

Active Networks evolve the packet processing pipeline. Specifically, they introduce the notion of *capsules*; network packets carrying data and processing code. On each hop, the default packet processing logic is extended with the capsule code. Active Networks provide user-driven protocol upgrades without any device or driver modification.

Active Network research defined two primary design aspects: the **capsule API and format** and the **switch architecture**. In terms of capsule format, the Active Network community defined the Active Network Encapsulation Protocol (ANEP) [Alexander et al., 1997a], adopted by the ANTS and Sprocket capsule programming frameworks. Sprocket [Schwartz et al., 2000] was developed by BBN technologies and defined a capsule programming language, based on C. Sprocket removed all insecure C structures, like pointers, and provided native support for SNMP browsing. Its compiler produced MIPS assembly code and the capsule code was executed using a MIPS VM on each networks device. Sprocket did not persist any state on the switch, but allowed capsule code to modify packet data. ANTS [Wetherall et al., 1998] was an attempt by the MIT Active Network group to develop a JAVA-based capsule programming environment. ANTS used a restricted version of the Java language, and switch and capsule integration was defined using Java interfaces. Capsule code could persist state on switches and modify packet content. An interesting functionality of the ANTS framework was the code dissemination mechanism. An end-node would deploy its protocol functionality solely to the local Internet gateway, and the code would be forwarded along the network towards the destination, hop-by-hop. Finally, the University of Pennsylvania's Active Network group developed PLAN [Hicks et al., 1998], an OCaml-based approach to capsule programming. PLAN did not follow the ANEP packet format. A PLAN capsule contained code and data, with the capsule code replacing the network header information. In order to secure switch infrastructure, the language disallowed packet data modification or switch state persistence. Ultimately, the language provided a framework for programmable control plane.

In terms of Active Network platform architecture, the community developed a number of architectures, addressing many efficiency aspects in capsule processing. The University of Pennsylvania developed the SwitchWare Execution Environment (EE) [Alexander et al., 1998a]. The architecture developed an OCaml capsule processing framework providing enhanced security for the switch and capsule execution environment. SwitchWare used SANE [Alexander et al., 1998b], a trustworthy operating system, to secure the functionality of the switch, and build on top of its primitives to provide higher level of security. The platform addressed issues regarding secure execution and authentication as well as capsule code verification. PLANet [Hicks et al., 1999] and Active Bridge [Alexander et al., 1997b] used the SwitchWare framework to

implement novel functionality in Active Network.

The Active Network group in the University of Arizona presented a switch architecture which optimized multi-layer packet processing using the communication-oriented Scout OS [Montz et al., 1995]. On top of Scout, the group developed the Liquid Software API [Hartman et al., 1999], providing a tight integration between the OS and the JVM, and improved capsule processing using the JIT JAVA compiler.

The CANEs project [Chae and Zegura, 2002] from the Georgia Tech Active Network group proposed a switch architecture, which improved flexibility in multi-protocol packet processing. Specifically, it defined a number of abstractions, which allowed multiple protocol stacking on the forwarding path. CANEs project build on top of the Bowman Switch OS [Merugu et al., 1999] and established a simple and efficient abstraction over switch resources. Researchers from Columbia University developed the NetScript switch programming language [da Silva et al., 2001], designed for flexible protocol processing definition and composition. The language provided seamless extensibility in protocol implementation logic, providing three types of protocol composition: layered composition; composition through protocol bridging; and end-to-end composition.

Finally, a joint effort between ETH and the University of St. Louis, developed a high performance capsule-enabled switch design. The High Performance Active Network Node (ANN) switch architecture [Decasper et al., 1999] used an FPGA-based CPU on each ATM interface to handle capsule execution. The CPU could run the ANTS EE and provided an IPv4 and IPv6 protocol processor supporting Gigabit rates.

Active Networks addressed a number of interesting problems in control plane functionality, especially issues like controllability and evolvability, and motivated recent efforts in the field. Nonetheless, proposed architectures followed a complex and clean-slate approach, and thus reduced their applicability in production environments. In addition, Active Network functionality benefited highly from the relatively low link rates of the time, which were manageable by the CPU chipsets. The exponential increase in link capacities, supporting up to Gigabit rates, restricts our ability to support the rich programmability of Active Network. Finally, the flexibility provided by Active Networks raised significant concerns on network security and resource controllability.

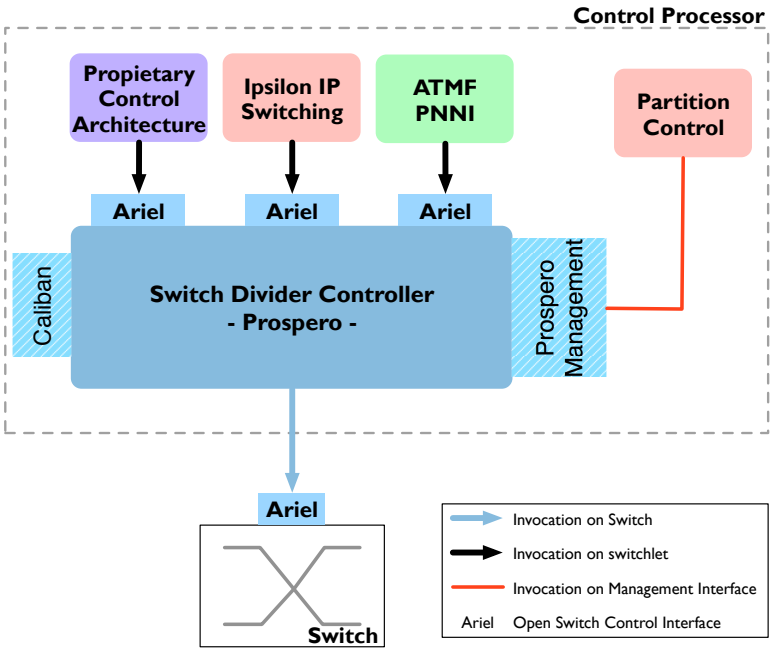


Figure 2.2: Tempest switch architecture [van der Merwe, 1998].

2.3.2 Devolved Control of ATM Networks

Active Networks focused extensively on Ethernet technologies and strived to develop protocol evolvability. A similar approach was developed in the University of Cambridge for the DCAN [DCAN project, 2000] project, focusing primarily on control evolvability and network virtualisation in ATM networks. DCAN tried primarily to simplify the control plane of ATM devices, in order to support network multi-functionality. Similar efforts were developed by IETF through the General Switch Management Protocol (GSMP) [Doria et al., 2002].

Rooney et al. [1998] presented Tempest, a novel ATM switch architecture, which enabled clean separation of the control and forwarding plane of an ATM switch. Similar to the SDN approach, the control plane was centralised in a single programmable entity, which could implement intelligent and dynamic forwarding. The implementation of Tempest was logically divided between three subsystems, depicted in Figure 2.2. The figure presents how a single switch is able to function in parallel as an IP router, an ATM switch and a Hollowman controller [Rooney, 1997], a devolved ATM

control framework.

Prospero Switch Divider The *Prospero* abstraction provides a mechanism for virtualising ATM switch resources into multiple virtual switches, called *switchlets*, through a resource control interface. A switchlet controls a subset of switch ports, VCI and VPI mappings, packet buffers and bandwidth. Prospero used the ATM QoS principles to implement packet buffer and bandwidth virtualisation. Fundamentally, Prospero is responsible for mapping the control interface, exposed to controllers, to the underlying switch functionality.

Ariel Switch Independent Control Interface Switchlets exposed forwarding control through the Ariel Interface. The Ariel Interface organises network control through six control objects: *Configuration*, *Port*, *Context*, *Connections*, *Statistics* and *Alarms*. The Configuration object provides details for the switch configuration; the Port object provides primitive controllability of ports (e.g. state, loopback functionality); the Context object enables QoS policy control; the Connection object exposes control of VPI/VCI mappings; the Statistics object exposes packet and byte counters; and the Alarms object pushes state change notifications to the controller. Tempest switches executes an Ariel server and translates Ariel requests to Prospero control requests. Ariel was fundamentally an abstraction layer between the switch silicon and the Prospero control interface.

Caliban Switch Management Interface In addition to network control, Tempest also supports evolved network management through the Caliban interface. The interface functionality is similar to the SNMP protocol. Caliban provides fine level SNMP-style information, as well as higher level aggregation operations over the switch state.

In addition to the redefinition of the network control abstraction, Tempest also proposed a relaxed network resource management scheme. Specifically, the architecture proposed a measurement-based admission control mechanism for circuit establishment [Lewis et al., 1998]. The measurement scheme redefined the static resource allocation scheme in an ATM network, and used effective bandwidth measurement techniques to estimate available resources and provide higher utilisation of network resources, with minimum relaxation of the QoS guarantees.

Tempest defined a highly efficient network control abstraction, which motivated modern network control approaches, like the SDN, to reimplement it over Ethernet devices. The simplicity of the control abstraction permitted integration of the technology with existing forwarding devices, and enabled line rate forwarding and efficient resource control. Nonetheless, its strong reliance on the ATM technology made the approach less relevant for the modern Ethernet-dominated networks [Crosby et al., 2002].

2.3.3 Software Defined Networking

SDN [Open Network Foundation, 2012], the most recent network control paradigm, provides a pragmatic approach to control evolvability. SDN employs a clean separation between the control and forwarding plane of a network device. The control logic is removed from the network device and implemented by a separate service. A well defined protocol establishes the integration between the two entities.

The SDN paradigm is motivated by relevant earlier attempts in network programmability(Active Network and DCAN), but follows an evolutionary approach. The paradigm is motivated by two key observations. Firstly, the evolution of computer networks has collapsed the layers of the OSI and TCP/IP models. Production networks use network devices (e.g. firewall, NAT, layer-5 switches) which operate on multiple layers of the network and engage extensively in layer violation. The SDN paradigm provides a unifying cross-layer control model which fits such functionality under a single interface. Secondly, although network complexity has increased, network devices still expose stand-alone, decentralised and inflexible configuration mechanisms (e.g. remote logic, CLI interfaces), and employ proprietary control protocols, which reduce device interoperability within the network. The SDN architecture establishes centralised and unified management, supporting reactive control. The OpenFlow protocol is currently the pre-dominant implementation of the SDN paradigm.

OpenFlow Protocol

The OpenFlow protocol was originally defined by the OpenFlow Consortium, an organisation of academic institutes, but currently the protocol development is steered by the ONF, a standards definition committee comprised of academic institutions , service

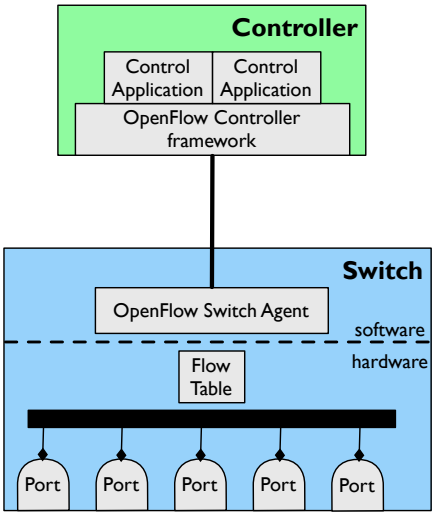


Figure 2.3: An elementary OpenFlow setup, consisting of a switch and a controller. implementating OpenFlow has motivated the definition of additional abstractions for control application and silicon adaption agents.

Field	OpenFlow Version		Field	OpenFlow Version
Src & Dst MAC addr.	1.0 [a]		input port	1.0
VLAN ID	1.0		VLAN PCP	1.0
IPv4 src/dst addr.	1.0		IPv4 ToS	1.0
ICMPv4 Type & Code	1.0		TCP/UDP/SCTP src/dst port	1.0
MPLS label	1.1		Metadata	1.1
MPLS class	1.1		IPv6 src/dst addr.	1.2
IPv4 proto	1.0		IPv6 flow label	1.2
ARP opcode	1.0		ICMPv6 type & code	1.2
ARP src/dst IPv4 address	1.0		ICMPv6 Network Discovery target address	1.2
ARP src/dst MAC address	1.2		ICMPv6 Network Discovery src/dst MAC address	1.2

[a] Since version 1.1 the protocol permits masked mac address matching

Table 2.1: OpenFlow tuple fields

Field	Operation	OpenFlow Version
OUTPUT	output packet to a port	1.0
SET_QUEUE	output packet to a port queue	1.0
SET_VLAN_VID	modify VLAN id	1.0
SET_VLAN_PCP	modify VLAN PCP	1.0
SET_DL_SRC SET_DL_DST	modify src/dst mac addr.	1.0
SET_NW_(SRC,DST)	modify IPv4 src/dst addr.	1.0
SET_NW_TOS	modify IPv4 ToS	1.0
SET_NW_ECN	modify IPv4 ECN bits	1.1
SET_TP_(SRC,DST)	modify TCP/UDP/SCTP src/dst port	1.0
COPY_TTL_(OUT,IN)	copy TTL value for IPv4 tunnels	1.1
SET_MPLS_LABEL	modify MPLS label	1.1
SET_MPLS_TC	modify MPLS traffic class	1.1
(SET,DEC)_MPLS_TTL	modify/decrement MPLS TTL	1.1
(PUSH,POP)_VLAN	Add/remove a VLAN header	1.1 [a]
(PUSH,POP)_MPLS	add/remove MPLS tag	1.1
(SET,DEC)_NW_TTL	modify/decrement IPv4 TTL value	1.1
(PUSH,POP)_PBB	remove/add a PBB service tag	1.3

[a] OpenFlow 1.0 defined a primitive to remove a VLAN header only

Table 2.2: OpenFlow packet actions

providers and network vendors. OpenFlow has been a highly successful approach to network control. It is readily available in a series of production devices as well as in large enterprise networks [Google, 2013; Kobayashi et al., 2014].

Figure 2.3 presents an elementary OpenFlow setup, consisting of two entities: the *controller* and the *switch*. The two entities communicate over a TCP *control channel*. The OpenFlow controller executes the network control logic and disseminates it to switches over the control channel. The controller is usually divided into two layers. A *controller framework* encapsulates the low level OpenFlow logic and exposes a higher level abstraction to the developer. Table 3.5 presents a complete list of available OpenFlow frameworks. Network control is implemented in *control applications*, running over the controller framework.

The OpenFlow switch, also called *datapath*, abstracts the control of a forwarding device. It consists of a set of *flow tables* and a set of *ports*. The flow table is a memory module containing flow entries which encode the active network policy. The data of a flow entry comprises of three structures: the flow tuple; the flow statistics; and the flow action list. The tuple defines packet matches using header fields to group packets into flows. The fields of the tuple have evolved over the years and Table 2.1 presents the tuple fields supported in each protocol version. Each tuple is associated with a field mask and permits wildcard matching. Flow statistics store packet and byte counters for each flow. The action list contains the list of actions applied to each matched packet. We present in Table 2.2 the packet processing actions defined by the OpenFlow protocol. The protocol defines a simple packet processing algorithm. For each packet, the datapath extracts a set of header fields and matches them against the entries of the flow tables. If a matching flow is found, then the flow statistics are updated and the action list is applied on the packet. If there is no matching entry in the flow table, then the packet is sent to the controller. The implementation of the OpenFlow switch abstraction in production switches is usually split between the hardware and software plane of the device. An OpenFlow agent runs on the switch co-processor, translating OpenFlow operations into an ASIC configuration.

The protocol defines a number of message types for controlling datapath resources. OpenFlow messages can be grouped into two categories: *forwarding control* and *switch configuration*. The rest of the section presents further details on available OpenFlow messages, and discusses the evolution of the protocol. We focus our protocol presen-

tation on version 1.0 of the protocol. ONF has released three revisions of the protocol, versions 1.1, 1.2 and 1.3, which significantly change the protocol specification. Nonetheless, the majority of production systems support version 1.0.

Forwarding control OpenFlow has two main control modes: *reactive* and *proactive* control. In reactive control, the switch forwards every unmatched packet to the controller, which is then responsible for responding with an appropriate modification in the flow table. The protocol defines the `pkt_in` message to encapsulate the header of a data plane packet when forwarded to the controller, and the `flow_mod` message to manipulate the flow table. The reactive control approach provides fine control over the traffic dynamics, but introduces significant load on the control plane. In proactive control, the controller is responsible for pre-installing all required flows in the flow table and avoiding any packet handling exceptions. The OpenFlow protocol provides the `flow_stats_req/flow_stats_resp`, `port_stats_req/port_stats_resp` and `aggr_stats_req/aggr_stats_resp` controller messages to poll for flow and port statistics and infer network resource utilisation. In order to ensure operational atomicity, the protocol provides the `barrier_req/barrier_reply` message to synchronise message execution between the controller and the switch. A `barrier_reply` is send by a switch as a response to a `barrier_req` from the controller, when all previously sent operations are processed. Finally, the OpenFlow protocol provides two additional message types to enhance control capabilities: the `pkt_out` and the `flow_removed`. The `pkt_out` message enables the controller to inject traffic in the data plane and the `flow_removed` message can notify the controller when a flow entry is removed from the flow table, due to timer expirartion or explicit removal by the user.

Switch configuration In addition to flow table control, the protocol provides switch configuration capabilities. A OpenFlow controller can use the `switch_config_req /switch_config_resp` message types, to discover the switch operational support of OpenFlow functionality. In addition, the protocol provides capabilities for controlling port states with the `port_mod` message type. The switch is also able to notify the controller when a port changes its state (e.g. link is detected to be inactive in the physical layer) with the `port_status` message. In recent revisions of the protocol, the

steering committee has introduced the capability for controlling per port traffic shaping queues.

Protocol evolution Since the specification of version 1.0 of the protocol, the OpenFlow steering committee has produced 3 non-backwardly compatible revisions. This protocol evolution is driven both by the osmosis between the hardware and software community and the introduction of new deployment use-cases.

Version 1.1 modifies the main protocol processing pipeline and exposes a switch abstraction which better matches the design of an ASIC. Specifically, the packet lookup process searches sequentially, instead of in parallel, between the switch flow tables. The packet must match an entry in each of the tables, otherwise a `pkt_in` message is generated. In addition, in order to persist partial results between tables, the protocol defines a 64-bit per-packet metadata field and the action list is augmented with operations over the metadata field and over the table search process (e.g. terminate lookup, skip table). Secondly, the protocol introduces a primitive to express multipath support using a new forwarding table, i.e. the group table. Group table entries consists of group entries containing flow actions and an assigned output port. A flow table entry can forward a matching packet to a group entry. The group selection action can forward a packet to all the buckets of the entry (ALL) or to a random bucket, similar to Equal Cost Multiple Path routing (ECMP) [Hopps, 2000] functionality; (SELECT) or select a specific bucket (INDIRECT); or to the first group entry with a non-blocked output port (FAST_FAILOVER). Thirdly, the protocol introduces MPLS support.

Version 1.2 of the protocol extends data plane protocol support. This revision introduces support for IPv6 and Provider Bridge Network (PBB - Mac-in-Mac) traffic. In addition, the protocol uses a flexible type-length-format (TLV) for flow tuple definitions. Finally, it defines a model for efficient multi-controller switch connectivity, in an effort to improve control channel resilience.

Version 1.3 introduces protocol support of QoS control. The protocol defines a new table abstraction, the metering table, which contains queue definitions. In addition, this protocol defines a control plane improvement using multiple parallel control channels to the controller, in order to parallelise `pkt_in` transmission, and defines a mechanism to support message fragmentation.

Because the protocol complexity has increased significantly in recent versions, the

Language	Controller
Python	NOX [Gude et al., 2008], POX [Pox, 2012], Pyretic [Monsanto et al., 2013]
C++	NOX [Gude et al., 2008]
JAVA	Maestro [Cai et al., 2011], Floodlight [Switch, 2013]
Haskell	Nettle [Voellmy et al., 2010]
C	Mul [Dipjyoti, 2013]
Javascript	Nodeflow [Berger, 2012]
Ruby	Trema [Trema, 2011]

Table 2.3: List of OpenFlow controllers, organised by programming language

Vendor	Model	OpenFlow version
HP	8200zl, 6600, 6200zl, 5400zl, and 3500/3500yl	v1.0
Brocade	NetIron CES 2000 Series	v1.0
IBM	RackSwitch G8264	v1.0
NEC	PF5240, PF5820	v1.0
Pronto	3290, 3780	v1.0
Juniper	Junos MX-Series	v1.0
Pica8	P-3290, P-3295, P-3780 and P-3920	v1.2

Table 2.4: List of hardware switches with OpenFlow support

vision for future OpenFlow functionality is to differentiate support between devices, and only optimize a subset of the functionality. For example, a firewall device can support only network and transport layer field match to improve flow table capacity.

2.4 SDN Applications

The SDN paradigm has proven extremely effective, both for the network community and the industry. A number of vendors provide production-level hardware SDN support (Table 2.4), while the SDN community provides OpenFlow support for the majority of programming languages (Table 2.3). In this section we iterate a series of network problem classes and present the design of a series of effective SDN applications which address these problems.

2.4.1 Network Virtualisation Applications

The introduction of OS virtualisation and the subsequent rise of the cloud computing paradigm, has created new opportunities for ICT to reduce infrastructure costs and provide unprecedented application resilience. Although OS virtualisation platforms, like Xen, provide high precision resource allocation and performance isolation, there is still a significant mismatch in the network abstraction. The introduction of the SDN abstraction in the datacenter provides novel opportunities to extend the virtualisation abstraction in network resources.

Sherwood et al. [2010a] presented FlowVisor, an early network virtualisation approach based on OpenFlow. FlowVisor is an OpenFlow proxy between switches and controllers, which transforms control over a set of switches into a single big virtual switch abstraction. FlowVisor employs a simple network topology discovery and resource control mechanism, while appropriate message processing allows to translate the view between the two abstractions. The administrator can partition the OpenFlow tuple, delegate network control to different controllers and, effectively, allow multiple tenants to exercise network control for their own network subnet over the same network infrastructure. FlowVisor quickly transformed into a network product by BigSwitch. Furthermore, Corin et al. [2012] developed a topology virtualisation extension for FlowVisor. FlowN [Drutskoy et al., 2013] follows a different approach and integrates virtualisation into the programming framework. The framework uses VLAN tags to multiplex data plane traffic and synchronizes controller states through an SQL database. Finally, Nicira provides the Nicira Network Virtualization Platform (NVP), a production level network virtualisation and control isolation framework. However, it is proprietary and its implementation details are undisclosed.

2.4.2 Security and Access Control Applications

SDN provides fast prototyping for secure network applications and unprecedented control reactivity to threat detection frameworks. Ethane [Casado et al., 2007], an early attempt to define the SDN abstraction, established a policy expression language to control interactions between user identities and network services. Ballard et al. [2010] focus on the problem of effective network monitoring and present OpenSAFE, a policy expression language for efficient traffic interception and inspection. Finally, Porras

et al. [2012] present FortNOX, an security extension for network virtualisation applications. FortNOX enables users to assign priorities and roles to control applications and automates policy inconsistency resolution.

2.4.3 Load Balancing Applications

The high popularity of network services with a global scope has introduced a requirement for novel indirection mechanisms, with high responsiveness and resilience. Current approaches employ proprietary load balancing devices, which dynamically distribute user requests between servers. SDN can integrate the indirection functionality in the network fabric. Wang et al. [2011] present an in-network proactive load balancing framework which uses the range of the OpenFlow wildcard matching to distribute client load between servers. Handigol et al. [2010, 2009] follow a reactive approach to the problem. For each new client, their Plug-and-Serve controller chooses on-the-fly a destination server, based on an allocation algorithm and the overall system load.

2.4.4 Inter-domain and Intra-domain Routing Applications

The SDN community has developed a series of applications to improve the performance of existing control plane protocols. Rothenberg et al. [2012] present an integration of the [Quagga, 2014] routing framework with the OpenFlow protocol, in an effort to revisit Route Reflectors in BGP routing [Bates et al., 2006]. Similarly, Kotronis et al. [2012] revisit the problem of BGP routing instabilities and proposes a framework for OpenFlow-based BGP calculation offloading to third parties.

The SDN has also motivated explorations on forwarding state compressibility. Sarrar et al. [2012] present a programming library for routing applications which exposes an FIB abstraction. The library at run-time analyses the FIB table and traffic pattern, and calculates the minimum set of flow entries which can serve a specific subset of the traffic on the fast-path of the network. Yu et al. [2010] present DIFANE, a valiant routing framework achieving significant FIB table compressibility. The proposed mechanism uses OpenFlow to partition effectively the network IP space and aggregate subsets of the FIB on each network device.

2.4.5 Network Management Applications

SDN control primitives can support the development of innovative network management frameworks. One of the first efforts was the Onix management framework [Koponen et al., 2010]. Onix provides a centralised control abstraction through its development API, which at run time was transparently distributed across the network.

The SDN paradigm has found significant applications in the formalisation of network management. Foster et al. [2011] presented Frenetic, a declarative programming language for network control policies, providing inherent support for significant network control programming requirements, like race conditions. Monsanto et al. [2012] developed NetCore, a policy expression language providing dynamic policy adaptation to traffic and topology changes and automatic optimization of the forwarding state. Guha et al. [2013] introduced verifiability to the NetCore language, while Monsanto et al. [2013] proposed an extension which provides seamless integration between control applications. Such control abstraction has also been discussed in the context of network upgradability. Reitblatt et al. [2012] present a verifiable framework which ensured network operability during updates. Finally, Voellmy et al. [2012] present a declarative policy language exposing a functional reactive programming model and providing flexible interface development.

2.4.6 Energy Control Applications

In recent years, energy consumption has become an important measure of the efficiency of a system's architecture. Currently the network is estimated to contribute approximately 20% of the total power consumption of a datacenter. Unlike CPUs, which can reduce power consumption when idle, network cards exhibit a constant high power consumption, due to continuous physical layer frame synchronisation. As a result the most efficient mechanism to reduce network power consumption is to shutdown the interfaces. Heller et al. [2010] present a power-aware control plane architecture, built on top of the OpenFlow abstraction. The application takes advantage of link redundancy and turns off interfaces when network utilisation is low.

2.4.7 Network Debugging and Measurement Applications

The SDN paradigm provides novel mechanisms for network troubleshooting and debugging. Wundsam et al. [2011] present OFRewind, a mechanism for intercepting and recording network policy inconsistencies for debugging purposes. Handigol et al. [2012b] present NDB, a control plane debugger which replicates the GDB abstraction in OpenFlow applications. NDB enables developers to register data plane breakpoint conditions and receive rich state information when these conditions are met. Finally, a number of applications have been proposed to enable control plane monitoring in order to detect potential network misconfigurations. Khurshid et al. [2012] present an OpenFlow proxy service which detects flow modifications that create forwarding policy inconsistencies. Canini et al. [2012] present a model-checking mechanism which uses symbolic execution to detect flow modifications that create significant inconsistencies in data plane forwarding policy. Recent efforts in remote network debugging have introduced OpenFlow-based solutions for home networks. Calvert et al. [2010] propose a network controller providing precise home network logging, in an effort to enable ISPs to troubleshoot home broadband connectivity problems.

2.4.8 Resource Control Applications

SDN reactivity enables fine-level resource control, and a number of applications have been developed for datacenter environments. Al-Fares et al. [2010] presented Hedera, a novel datacenter control architecture, providing better network utilisation in comparison to the ECMP [Hopps, 2000] network load-balancing mechanism. Hedera uses OpenFlow statistics to discover progressively network-limited flows and reroute them over underutilised network paths. Benson et al. [2011] presented a network load distribution mechanism designed to exhibit fairness towards short-lived flows. Even et al. [2012] employ OpenFlow as a network embedding mechanism for datacenter job assignments and to provide strong resource guarantees.

Resource control has also been proposed in the context of home networks. Yiakoumis et al. [2011] propose home network virtualisation in an effort to improve resource allocation on the edges, and enable core infrastructure sharing between ISPs. Furthermore, Yiakoumis et al. [2012] evolve this idea and introduce a simple user interface which can translate user resource requirement into ISP-wide network policies.

2.4.9 Network Mobility Applications

The SDN paradigm provides novel control capabilities in mobile networks. Huang et al. [2010] present PhoneNet, a mobile application development framework which provides the ability for dynamic multicast groups setup. Yap et al. [2009, 2010] present OpenRoads, a network control plane for wireless networks, enabling seamless Access Point (AP) hand-overs and network virtualisation. Finally, Li et al. [2012] discuss SDN applicability for cellular networks.

2.4.10 Network Experimentation

Currently, a number of large-scale shared experimentation infrastructures provide OpenFlow support. The GENI testbed [Global Environment for Network Innovations (GENI), 2013] funded by the NSF, and the Ofelia testbed [Ofelia, 2013] funded by the EU, FP7 framework provide network and computing resources for running cost-free large-scale experiments. Erickson et al. [2011] present Virtue, a large-scale Xen-based emulation framework which uses OpenFlow to enable scalable user experimentation with network topologies, and VM placements in a multi-tenant data center environment. Similarly, Mininet Handigol et al. [2012a] provides a scalable platform for OpenFlow experimentation and reproducibility. Mininet uses the LXC [LXC, 2013] Linux user-space virtualisation framework and network namespaces to run large scale topologies in a single host and experiment with network functionality. Mininet has been used in the Stanford Computer Science department to introduce students to recent research efforts [McKeown et al., 2012].

2.5 Summary

In this section we provided an in-depth analysis of available network control mechanisms. We started the discussion by presenting a generic architectural model for network devices, and highlighted the inherent limitations of network control. Furthermore, we presented the current production control mechanisms and, motivated by their limitations, we presented three experimental network control frameworks, i.e. Active Network, Devolved Control for ATM Network and Software Defined Networking. Finally, we provided an extensive presentation of recent efforts to improve modern

network functionality through control plane redesign. In the next chapter we present an extensive study on the scalability of the SDN paradigm.

Chapter 3

SDN Control Plane Scalability

This dissertation contends that control plane applications can scale aspects of computer network functionality. In our exploration we extensively employ the SDN design paradigm and the OpenFlow abstraction, primarily due to two core functional properties. Firstly, the clean separation of the network control and data plane is inherently backwards-compatible with existing network applications and interoperable with existing network devices. The evolution of the control plane of a network does not affect data plane protocol support and allows progressive deployment in production networks. Secondly, the OpenFlow control abstraction is sufficiently generic to support various control approaches. The protocol supports reactive and proactive control schemes, while the flow definition granularity can be dynamically controlled to match the environment requirements. The following chapters of this thesis employs OpenFlow to address two diverse network scalability problems.

This chapter presents an extensive scalability analysis of available SDN technologies. Our exploration provides an in-depth analysis of the limitations of existing implementation efforts and their impact on data plane performance. The work focuses on implementations of version 1.0 of the OpenFlow protocol, the predominant production-level protocol instantiation of SDN. We present two measurement platforms: OFLOPS and SDNSIM. OFLOPS is a high precision OpenFlow switch micro-benchmark platform. Using OFLOPS, we implement a set of benchmark tests and characterise the performance of elementary protocol interactions. SDNSIM is a macro-benchmark OpenFlow platform, which extends the Mirage framework [Madhavapeddy et al., 2013] to support large scale OpenFlow-based network simulations and emulations. Using

SDNSIM, experimenters can import OFLOPS switch profiles into their experiment and test the performance of their SDN design.

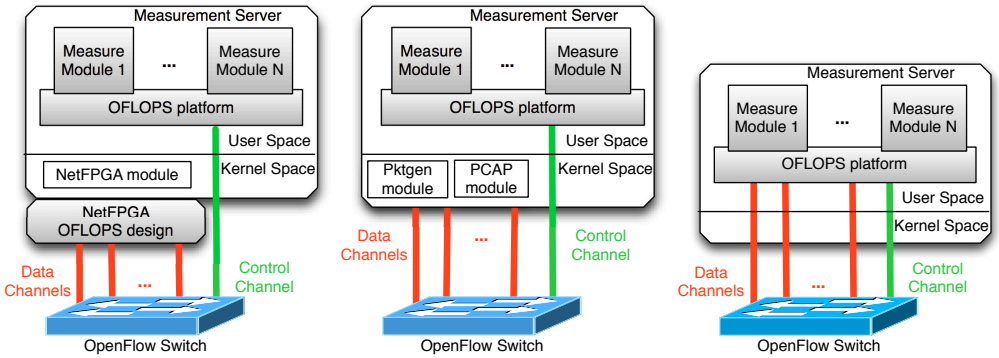
In this Chapter, we present the motivations (Section 3.1) and the design overview of OFLOPS (Section 3.2). We select a number of off-the-self OpenFlow switches (Section 3.3) and assess the elementary protocol interaction performance (Section 3.4). Furthermore, we discuss some of limitations in network experimentation and present SDNSIM (Section 3.5) and its architecture (Section 3.6). Finally, we assess the performance scalability and fidelity of SDNSIM (Section 3.7), along with a measurement study of control scalability over the fat-tree topology (Section 3.8), and conclude our work (Section 3.9).

3.1 Network Control Micro-benchmark

Despite the recent introduction of the SDN paradigm, the research community has already proposed many novel control architectures, presented extensively in Section 2.4. These architectures address significant problems of modern networking, but their deployment in production environments is not straightforward. Computer networks have become a vital asset for modern enterprises. High availability and performance are critical and any control plane modification must fulfil these criteria, requiring extensive testing and performance characterisation. Weissmann and Seetharaman [2011] present the deployment experience of the first OpenFlow network in the Computer Science department of Stanford University, and report significant performance and reliability problems. The performance limitations were significantly influenced by the inability of the hardware switching platform to process OpenFlow interactions at high rates. Fundamentally, the SDN development toolchain lacked generic tools to assess OpenFlow scalability. Traditional switch performance models use metrics like packet latencies and MAC table size. The control decoupling of the SDN paradigm expands the interaction freedom between a controller and a switch device, and effectively increases the complexity of characterising switch performance.

In order to fulfil the requirement for OpenFlow switch performance evaluation, we developed OFLOPS[1], a measurement framework enabling rapid development of

[1]OFLOPS is under the GPL licence and can be downloaded from http://www.openflow.org/wk/index.php/Oflops



(a) NetFPGA packet (b) Kernel-space (c) User-space

Figure 3.1: OFLOPS enables the development of custom multi-channel measurement experiments and supports: 1) the high accuracy NetFPGA packet generator hardware design (Figure 3.1(a)), 2) the kernel space `pktgen` traffic generation module and the PCAP packet capturing module (Figure 3.1(b)), 3) a userspace packet capture and generation library (Figure 3.1(c)).

performance tests for hardware and software OpenFlow switch implementations. To better understand the behaviour of the tested OpenFlow implementations, OFLOPS combines OpenFlow control and data plane measurements. To ensure sub-millisecond-level accuracy of the measurements, we bundled the OFLOPS software with specialized hardware in the form of the NetFPGA platform[1]. Note that if the tests do not require millisecond-level accuracy, commodity hardware can be used instead of the NetFPGA [Arlos and Fiedler, 2007].

3.2 OFLOPS Design

Measuring OpenFlow switch implementations is a challenging task in terms of the characterization of accuracy, noise suppression and precision. Performance characterization is non-trivial, as most OpenFlow-enabled devices provide rich functionality but do not disclose implementation details. In order to understand the performance impact of an experiment, multiple input measurement channels must be monitored concurrently, such as data and control channels. Furthermore, current controller frameworks, like Gude et al. [2008]; Switch [2013], are designed for production networks and in-

[1] http://www.netfpga.org

cur significant measurement noise. Such controllers employ asynchronous processing libraries and multi-thread execution models, to maximize the aggregate control channel processing throughput, and provide high-level programming interfaces that require multiple data structure allocation and modification during packet processing. The processing time for a specific OpenFlow message, especially at high rates, is subject to multiple aspects, like thread scheduling policy and asynchronous library execution logic [Jarschel et al., 2012]. Measurement noise suppression in the control plane is required from the controller framework to minimize message processing and delegate processing control to the measurement module. Finally, sub-millisecond precision in software is subject to unobserved parameters, like OS scheduling and clock drift. The result of these challenges is that meaningful, controlled, repeatable performance tests are non-trivial in an OpenFlow environment.

OFLOPS has a low overhead abstraction, enabling interaction with an OpenFlow-enabled device over multiple data channels. The platform provides a programming API, allowing control of multiple information sources: data and control channels, and SNMP, following an event-driven architecture. Interactions over each channel are transformed into semantically meaningful events and the experimenter can register callbacks to handle them programatically. OFLOPS is implemented in C, and experiments are compiled as shared libraries, loaded at run-time using a simple configuration interface. A schematic of the platform is presented in Figure 3.2, while OFLOPS [2011] provides further details regarding the OFLOPS programming model.

The platform is implemented as a multi-threaded application and takes advantage of modern multicore environments. To reduce latency, our design avoids concurrent access controls: we leave any concurrency-control complexity to individual module implementations. OFLOPS processing occurs in the following five threads:

1. Data Packet Generation: control of data plane traffic generators.

2. Data Packet Capture: data plane traffic interception.

3. Control Channel: controller events dispatcher.

4. SNMP Channel: SNMP event dispatcher.

5. Time Manager: time events dispatcher.

OFLOPS can run on heterogeneous platforms and integrates support support for multiple packet generation and capturing mechanisms. Packet generation functionality is supported through three mechanisms: userspace (Figure 3.1(c)); kernel-space

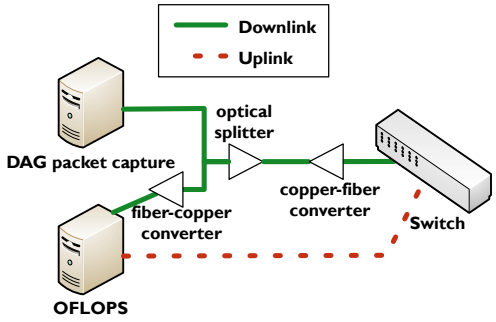


Figure 3.2: OFLOPS precision evaluation topology against a DAG card.

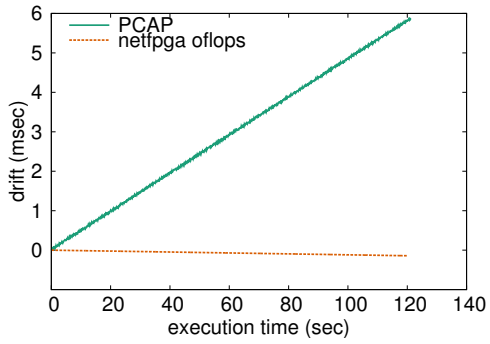


Figure 3.3: Timestamping precision evaluation using a DAG card. PCAP timestamps exhibit a significant drift in the order of microseconds, in comparison to the NetFPGA hardware design.

using the pktgen kernel module module [Olsson, 2005] (Figure 3.1(b)); and hardware-accelerated through a modified version of the NetFPGA Stanford Packet Generator [Covington et al., 2009] (Figure 3.1(a)). OFLOPS supports both the PCAP library and the modified NetFPGA design for packet capturing and timestamping. Each mechanism provides different precision guarantees.

Figure 3.2 presents a simple evaluation setup for the accuracy of the OFLOPS capturing timestamps, in comparison to a DAG card [Endace, 2012]. For the measurement, we use a 100 Mbps CBR probe of small packets (90 bytes) for a two minute period. The probe is duplicated, using an optical wiretap with negligible delay, and sent simultaneously to OFLOPS and to the DAG card. In Figure 3.3, we plot per-packet drift between each OFLOPS timestamping mechanism and the DAG card. From the figure, we note that the PCAP timestamps drift by 6 ms after 2 minutes, while the NetFPGA timestamping mechanism exhibits lower drift on the order of microseconds.

3.3 Measurement Setup

At the end of 2009, the OpenFlow specification was released in its first stable version, 1.0 [OpenFlow Consortium, 2009b], the first recommended version implemented by vendors for production systems. Our analysis was conducted in collaboration with T-labs in Spring of 2011. During that period, the introduction of OpenFlow support

in commodity switches was limited, and only a small number of devices provided production-level support for the protocol. Using OFLOPS, we evaluated OpenFlow-enabled switches from three different switch vendors. Vendor 1 provided production-ready OpenFlow support, whereas vendors 2 and 3 during that period provided experimental OpenFlow support. The set of selected switches provided a representative but not exhaustive sample of available OpenFlow-enabled ToR-type switching hardware, during that time. Finally, we also include in our measurement a recent OpenFlow switch which provides production support up to version 1.1 of the protocol (firmware was released in December 2013). Switch4 uses the Open vSwitch codebase for its implementation. Table 3.1 provides details regarding the CPU and the size of the flow table for each switch. In addition, the switching fabric in the vendor hardware specification reports similar non-blocking capacity and packet processing rates.

OpenFlow is not limited to hardware. The protocol reference implementation is the software switch, Open vSwitch [Pettit et al., 2010], an important implementation for production environments. Firstly, Open vSwitch provides a replacement for the poor-performing Linux bridge [Bianco et al., 2010], a crucial functionality for virtualised operating systems. Currently, the Xen platform uses Open vSwitch to forward traffic between VMs in the Dom0, a configuration which is inherited by major Cloud providers, like Amazon and Rackspace. Secondly, several hardware switch vendors use the Open vSwitch codebase for the development of their own OpenFlow-enabled firmwares. The Open vSwitch development team has standardised a clean abstraction for the control of packet forwarding elements (similar to Linux HAL), which allows code reuse by any forwarding entity. Thus, the mature software implementation of the OpenFlow protocol is ported to commercial hardware, making certain implementation bugs less likely to (re)appear. We preclude Open vSwitch in our performance and scalability study of hardware switches. Finally, in our comparison we include the OpenFlow switch design for the NetFPGA platform [Naous et al., 2008]. This implementation is based on the original OpenFlow reference implementation [OpenFlow Consortium, 2009a], extending it with a hardware forwarding design.

In order to conduct our measurements, we setup OFLOPS on a dual-core 2.4GHz Xeon server equipped with a NetFPGA card. For all of the experiments we utilized the NetFPGA-based packet generating and capturing mechanism. 1 GbE control and data channels are connected directly to the tested switches. We measured the processing

Switch	CPU	Flow table size
Switch1	PowerPC 500MHz	3072 mixed flows
Switch2	PowerPC 666MHz	1500 mixed flows
Switch3	PowerPC 828MHz	2048 mixed flows
Switch4	MIPS 1.1 SoC	2048 mixed flows
Open vSwitch	Xeon 3.6GHz	1M mixed flows
NetFPGA	DualCore 2.4GHz	32K exact & 100 wildcard

Table 3.1: OpenFlow switch details.

delay incurred by the NetFPGA-based hardware design to be a near-constant 900 ns.

3.4 Switch Evaluation

As for most networking standards, there are different ways to implement a given protocol based on a paper specification. OpenFlow is no different in this regard. The current OpenFlow reference implementation is Open vSwitch [Pettit et al., 2010]. However, different software and hardware implementations may not implement all features defined in the Open vSwitch reference, or they may behave in an unexpected way. In order to understand the behaviour of an OpenFlow switch implementation, we developed a suite of measurement experiments to benchmark the functionality of the elementary protocol interactions. These tests target:

1. The OpenFlow packet processing actions (Section 3.4.1)
2. The packet interception and packet injection functionality of the protocol (Section 3.4.2)
3. The update rate of the flow table and its impact on the data plane, (Section 3.4.3)
4. The monitoring capabilities provided by OpenFlow (Section 3.4.4)
5. The impact of interactions between different OpenFlow operations (Section 3.4.5).

3.4.1 Packet Modifications

The OpenFlow 1.0 specification [OpenFlow Consortium, 2009b] defines 10 packet modification actions which can be applied on incoming packets. Available actions include modification of source and destination MAC and IP addresses, VLAN tag and PCP fields and TCP and UDP source and destination port numbers. The action list of a flow definition can contain any combination of them. The left column of Table 3.2 lists the packet fields that can be modified by an OpenFlow-enabled switch. These actions are used by network devices such as IP routers (e.g. rewriting of source and destination MAC addresses) and NAT (rewriting of IP addresses and ports). Existing network equipment is tailored to perform a subset of these operations, usually in hardware to sustain line rate.

To measure the time taken by an OpenFlow switch to modify a packet field header, we generated from the NetFPGA card UDP packets of 100 bytes, at a constant rate of 100Mbps (approximately 125 Kpps). This rate is high enough to give statistically significant results in a short period of time, without introducing packet queuing in software-based switches. The flow table is initialized with a flow that applies a specific action on all probe packets and the processing delay is calculated as the difference between the transmission and receipt timestamps provided by the NetFPGA card. We report in Table 3.2 the median processing delay for each action, along with its standard deviation, and the percent of lost packets of the measurement probe.

We observe significant differences in the performance of the hardware switches due in part to the way their firmware implements packet modifications. Switch1 and Switch2, with their production-grade implementations, handle all modifications in hardware; this explains their low packet processing delay of between 2 and 4 μs. On the other hand, Switch2 and Switch3 each run experimental firmware that provided, at the time, only partial hardware support for OpenFlow actions. Switch2 uses the switch CPU to perform some of the available field modifications, resulting in processing delay and variance two orders of magnitude higher packet. Switch3 follows a different approach; all packets of flows with actions not supported in hardware are silently discarded. The performance of the Open vSwitch software implementation lies between Switch1 and the other hardware switches. Open vSwitch fully implements all OpenFlow actions. However, hardware switches outperform Open vSwitch as the flow

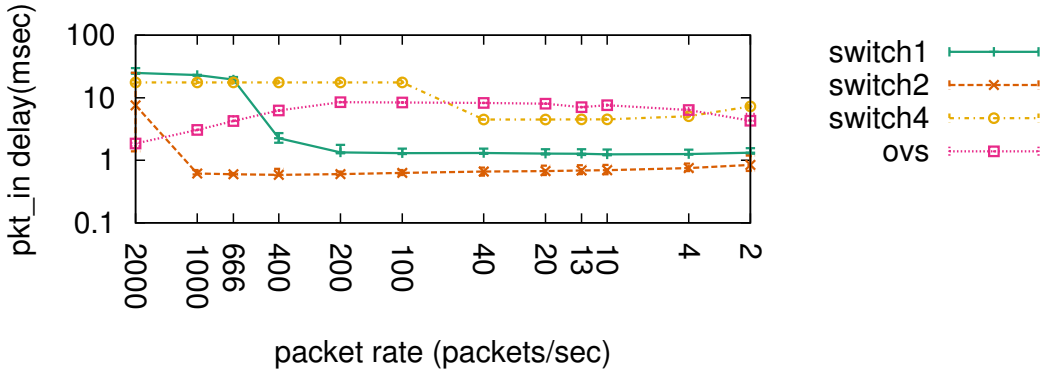
Mod. type	Switch1			Switch2			Switch3		
	med	sd	loss%	med	sd	loss%	med	sd	loss%
Forward	4	0	0	6	0	0	5	0	0
MAC addr.	4	0	0	302	727	88	-	-	100
IP addr.	3	0	0	302	615	88	-	-	100
IP ToS	3	0	0	6	0	0	-	-	100
L4 port	3	0	0	302	611	88	-	-	100
VLAN pcp	3	0	0	6	0	0	5	0	0
VLAN id	4	0	0	301	610	88	5	0	0
VLAN rem.	4	0	0	335	626	88	5	0	0

Mod. type	Switch4			Open vSwitch			NetFPGA		
	med	sd	loss%	med	sd	loss%	med	sd	loss%
Forward	2	0	0	35	13	0	3	0	0
MAC addr.	2	0	0	35	13	0	3	0	0
IP addr.	2	0	0	36	13	0	3	0	0
IP ToS	2	0	0	36	16	0	3	0	0
L4 port	2	0	0	35	15	0	3	0	0
VLAN pcp	2	0	0	36	20	0	3	0	0
VLAN id	2	0	0	35	17	0	3	0	0
VLAN rem.	2	0	0	35	15	0	3	0	0

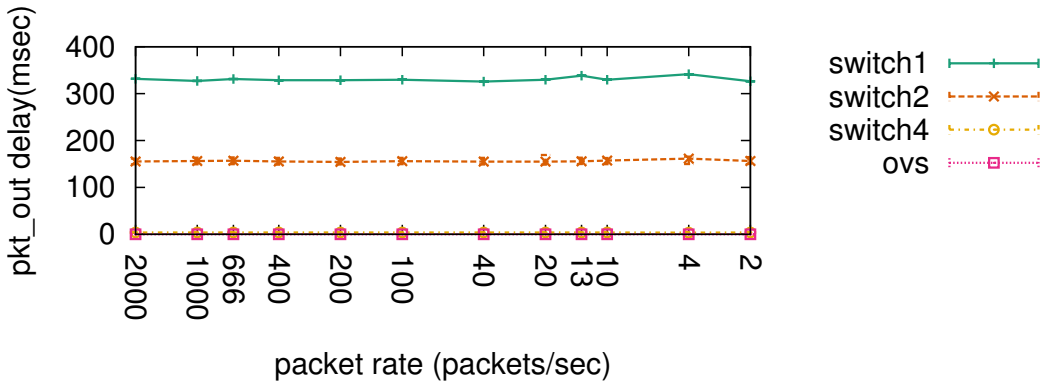
Table 3.2: Time in μs to perform individual packet modifications and packet loss. Processing delay indicates whether the operation is implemented in hardware ($<10\mu$s) or performed by the CPU ($>10\mu$s).

actions are supported in hardware. Finally, the NetFPGA design provides support for all actions in hardware and allows line rate support with low and constant processing latency.

We conducted a further series of experiments with variable numbers of packet modifications in the flow action list. We observed, that the combined processing time of a set of packet modifications is equal to the highest processing time across all individual actions in the set (e.g. Switch2 required approximately 300 ms per packet to modify both IP source addresses and IP ToS field). Furthermore, we noticed that Switch1, Switch4 and Open vSwitch have a limit of 7 actions, which exposes limits enclosed in the implementation.



(a) `pkt_in` processing latency



(b) `pkt_out` processing latency

Figure 3.4: Latency to intercept (Figure 3.4(a)) and inject (Figure 3.4(b)) data plane packets using the OpenFlow protocol.

3.4.2 Traffic Interception and Injection

OpenFlow protocol permits a controller to intercept or inject traffic through the control plane. Packet interception is fundamental for reactive control, while packet injection enables applications to interact with data plane end-hosts and devices, e.g. topology discovery using the LLDP protocol. In order to characterise the performance of these functionalities, we conducted two experiments. The first experiment invalidates all flow table entries and uses a data plane probe of small packets (100 bytes) on a data channel to measure the per-packet delay between the packet transmission and its reception on the control plane, as a `pkt_in` message. The second experiment generates a control plane probe using `pkt_out` messages and measures the delay in receiving

each packet on a data channel.

Figure 3.4 presents the 10_{th}, 90_{th} percentiles and median per-packet processing latency for varying probe rates. We omitted Switch3 in this experiment, because these functionalities incurred significant co-processor load, resulting in control plane unresponsiveness. In terms of packet injection, hardware switches exhibit high latency (150 ms for Switch2 and 350 ms for Switch1), in comparison to Open vSwitch and Switch4 (approximately 0.1 ms). For hardware switches, we noticed at high packet rates that packet transmissions were rate-limited using the advertised window of the control plane TCP connection. In terms of traffic interception, we observed diverse behaviours between hardware switches at high packet rates. For Switch1, latency becomes note-worthy beyond 400 packets/second, resulting in a median latency of 1 ms, while the switch can process up to 500 packets/second, with median latency of 10 ms. For Switch2, latency is significantly lower, with a median value of 800 μs and reaches 10 seconds only when the probe rate is 2,000 packets/second. Switch4 exhibits a constant delay, between 9 and 17 ms, but the packet rate is limited to 50 packets/second. Open vSwitch, has a high but stable latency, between 1 and 10 ms, for any tested data rate.

3.4.3 Flow Table Update Rate

The flow table is a central component of an OpenFlow switch and is the equivalent of a Forwarding Information Base (FIB) on routers. Given the importance of FIB updates on commercial routers, e.g. to reduce the impact of control plane dynamics on the data plane, the FIB update processing time of commercial routers provides useful reference points and lower bounds for the time to update a flow entry on an OpenFlow switch. The time to install a new entry on commercial routers has been reported in the range of a few hundreds of microseconds [Shaikh and Greenberg, 2001].

OpenFlow has a mechanism for defining barriers between sets of commands: the `barrier` command. According to the OpenFlow specification [OpenFlow Consortium, 2009b], the `barrier` command is a way of being notified that a set of OpenFlow operations has been completed. Furthermore, the switch has to complete the set of operations issued prior to the `barrier` before executing any further operation. If the OpenFlow implementations comply with the specification, we expect to receive a

`barrier` notification for a flow modification once the flow table of the switch has been updated, implying that the change can be seen from the data plane.

We checked the behaviours of the tested OpenFlow implementations, finding variation among them. For Open vSwitch, and Switch1, Figure 3.6 shows the time to install a set of entries in the flow table. Switch2 and Switch3 are not reported, as this OpenFlow message is not supported by the firmware. For this experiment, OFLOPS worked on a stream of packets of 100 bytes at a constant rate of 100 Kpackets/second (10 Mbps) that targeted the newly installed flows in a round-robin manner. The probe achieved sufficiently low inter-packet periods in order to accurately measure the flow insertion time.

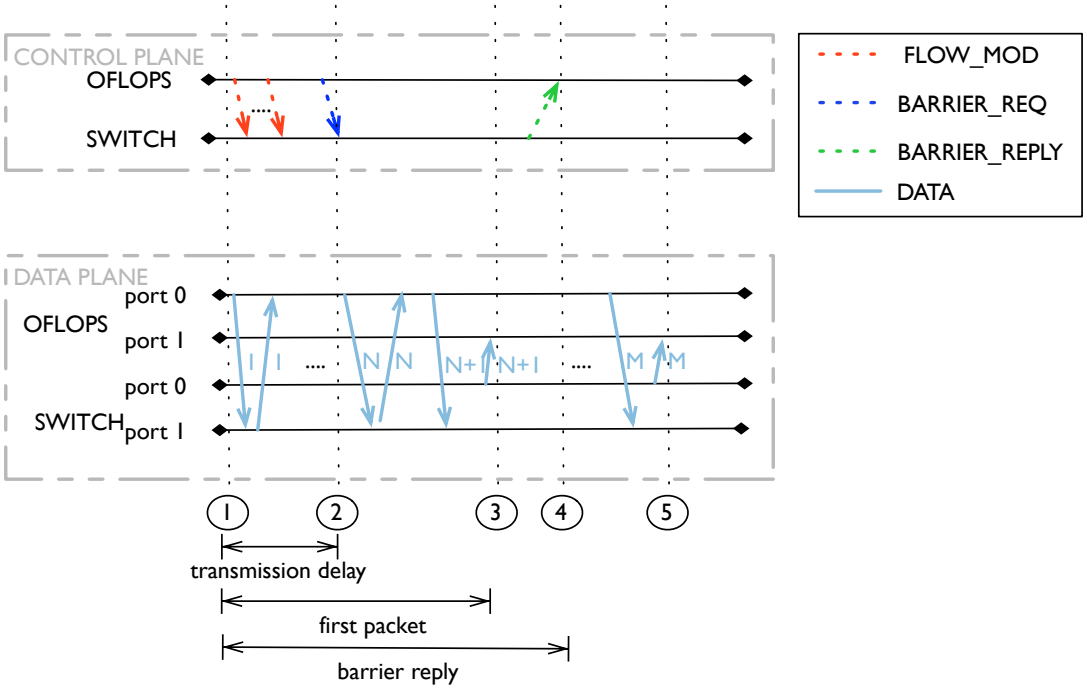
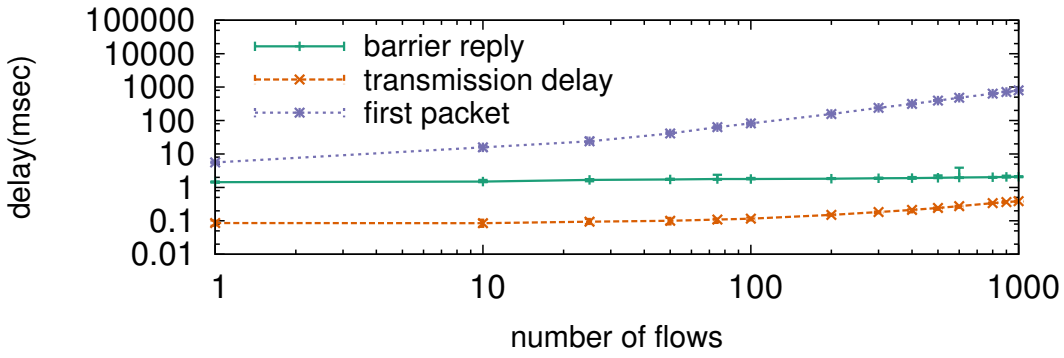
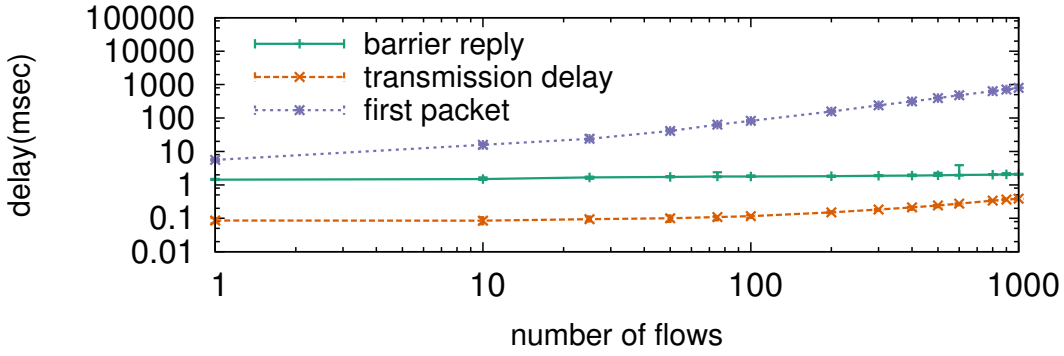


Figure 3.5: Flow insertion and flow modification measurement scenario. The experiment uses multiple information channels to estimate the delay in modifying entries in the flow table.

Figure 3.4.3 presents the measurement scenario, while 3.6 presents three different measured latencies. The first, *barrier reply* is derived by measuring the time elapsed between when the **first insertion command** is sent by the OFLOPS controller and the time the `barrier` reply is received by the PC. The second, *transmission delay*, is the



(a) Open vSwitch (log-log scale)



(b) Switch1 (log-log scale)

Figure 3.6: Flow entry insertion delay: as reported using the `barrier` notification and observed at the data plane.

time elapsed between when the first and last flow insertion commands are sent out from the PC running OFLOPS. The third, *first packet*, is the time elapsed between when the **first insertion command** is issued and when a packet has been observed for the last of the (newly) inserted rules. For each configuration, we ran the experiment 100 times and Figure 3.6 shows the median result as well as the 10^{th} and 90^{th} percentiles, although the variations are small and cannot be easily viewed.

From Figure 3.6, we observe that even though the *transmission delay* in sending flow insertion commands increases proportionally to the flow number, this time is negligible when compared with data plane measurements (*first packet*). Notably, the *barrier reply* measurements are almost constant, increasing only as the transmission delay

increases (difficult to discern on the log-log plot) and, critically, this operation returns before any *first packet* measurement. This implies that the way the *barrier reply* is implemented does not reflect the time when the hardware flow-table has been updated.

These results demonstrate how OFLOPS computes per-flow overheads. We observed that the flow insertion time for Switch1 starts at 1.8 ms for a single entry, but converges toward an approximate overhead of 1 ms per inserted entry as the number of insertions grows.

Flow insertion types

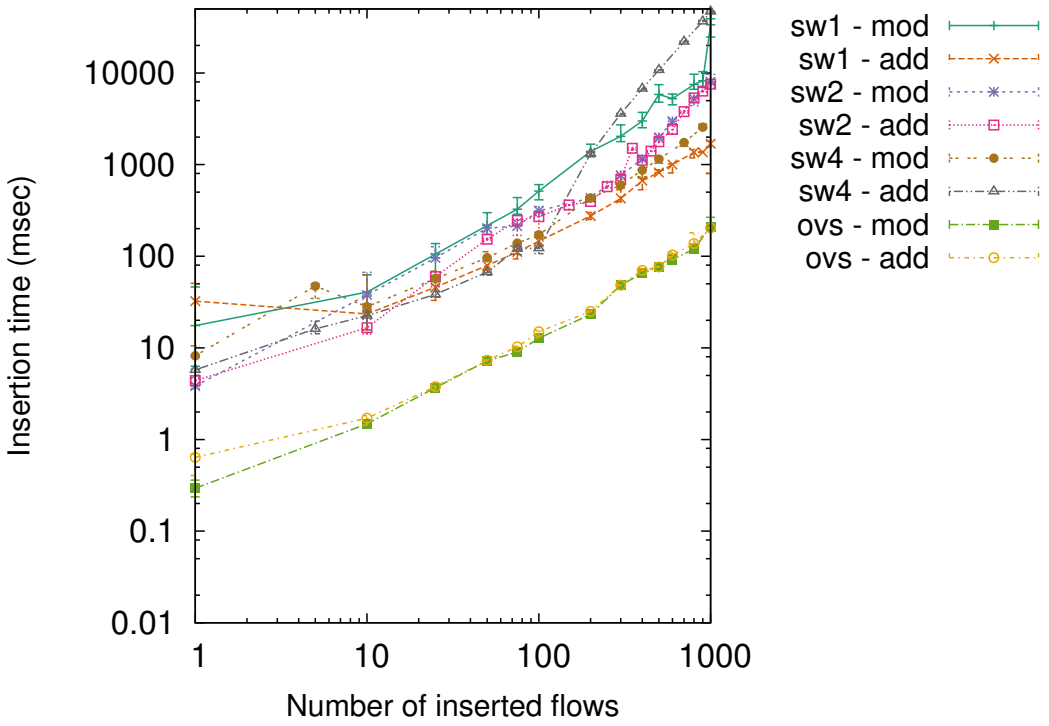


Figure 3.7: Delay of flow insertion and flow modification, as observed from the data plane (log-log scale).

We now distinguish between flow insertions and the modification of existing flows. With OpenFlow, a flow rule may perform exact packet matches or use wildcards to match a range of values. Figure 3.7 measures the flow insertion delay as a function of

the number of inserted entries. This is done for the insertion of new entries and for the modification of existing entries.

These results show that for software switches that keep all entries in memory, the type of entry or insertion does not make a difference in the flow insertion time. Surprisingly, both Switch1 and Switch2 take more time to modify existing flow entries compared to adding new flow entries. For Switch1, this occurs for more than 10 new entries, while for Switch2 this occurs after a few tens of new entries. After discussing this issue with the vendor of Switch2, we came to the following conclusion: as the number of TCAM entries increases, updates become more complex as they typically require re-ordering of existing entries. Gupta and McKeown [2001] characterise the update complexity of a TCAM as linear.

Clearly, the results depend both on the entry type and implementation. For example, exact match entries may be handled through a hardware or software hash table. Whereas wildcarded entries, requiring support for variable length lookup, must be handled by specialized memory modules, such as a TCAM. With such possible choices and a range of different experiments, the flow insertion times reported in Figure 3.7 are not generalizable, but rather depend on the type of insertion entry and implementation.

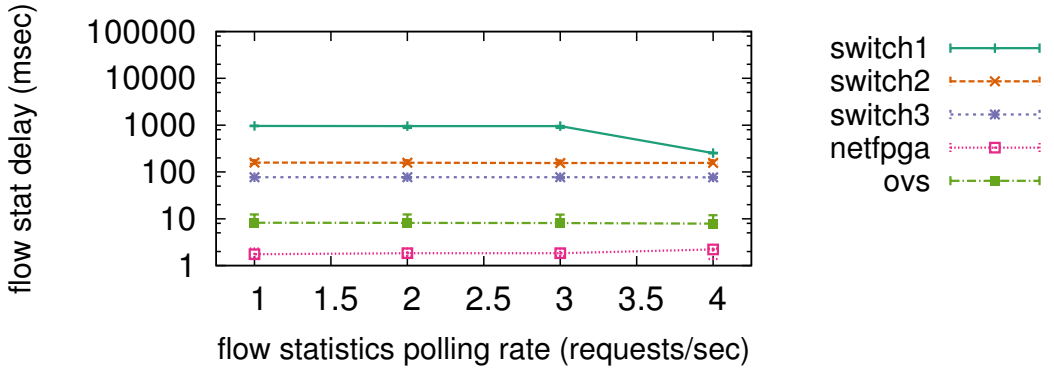
3.4.4 Flow Monitoring

The use of OpenFlow as a monitoring platform has already been suggested for the applications of traffic matrix computation [Balestra et al., 2010; Tootoonchian et al., 2010] and identifying large traffic aggregates [Jose et al., 2011]. To obtain direct information about the state of the traffic received by an OpenFlow switch, the OpenFlow protocol provides a mechanism to query traffic statistics, either on a per-flow basis or across aggregates matching multiple flows and supports packet and byte counters.

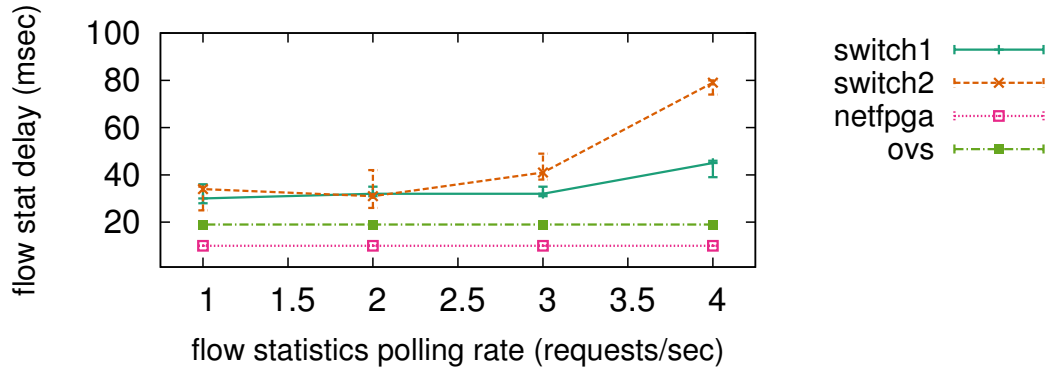
We now test the performance implications of the traffic statistics reporting mechanism of OpenFlow. Using OFLOPS, we installed flow entries that match packets sent on the data path. Simultaneously, we started sending flow statistics requests to the switch. Throughout the experiment we recorded the delay in getting a reply for each query, the amount of packets that the switch sends for each reply, and the departure and arrival timestamps of the probe packets.

Figure 3.8(a) reports the time taken to receive a flow statistics reply for each switch

as a function of the request rate. Despite the rate of statistics requests being modest, quite high CPU utilization is recorded for even a few queries per second being sent. Figure 3.8(b) reports the switch-CPU utilization as a function of the flow statistics inter-request time. Statistics are retrieved using SNMP. Switch3 is excluded for lack of SNMP support.



(a) Reply time.



(b) CPU utilization.

Figure 3.8: Time to receive a flow statistic (median) and corresponding CPU utilization.

From the flow statistics reply times, we observe that all switches have (near) constant response delays; the delay itself relates to the type of switch. As expected, software switches have faster response times than hardware switches, reflecting the availability of the information in memory without the need to poll multiple hardware counters. These consistent response times also hide the behaviour of the exclusively

hardware switches, whose CPU time increases proportionally with the rate of requests. We observe two types of behaviour from the hardware switches: the switch has a high CPU utilization, answering `flow_stats` requests as fast as possible (Switch2); or the switch delays responses, avoiding over-loading its CPU (Switch1). Furthermore, for Switch1, we noticed that the switch is applying a pacing mechanism on its replies. Specifically, at low polling rates the switch splits its answer across multiple TCP segments, each segment containing statistics for a single flow. As the probing rate increases, the switch will aggregate multiple flows into a single segment. This suggests that independent queuing mechanisms are used for handling flow statistics requests. Finally, neither software nor NetFPGA switches see an impact of the `flow_stats` rate on their CPU, thanks to their significantly more powerful PC CPUs (Table 3.1).

3.4.5 OpenFlow Command Interaction

An advanced feature of OpenFlow is its ability to provide network control applications with, for example, flow arrival notifications from the network, while simultaneously providing fine-grain control of the forwarding process. This permits applications to adapt in real time to the requirements and load of the network [Handigol et al., 2009; Yap et al., 2009]. Using OFLOPS measurement instrumentation, we developed a test scenario of dynamic network control, in order to understand the behaviour of the switch control and data plane. In this scenario, we emulated the simultaneous querying of traffic statistics and modification of the flow table. More specifically, we extended Section 3.4.3 by showing how the mechanisms of traffic statistics extraction and table manipulation interact. Specifically, we initialized the flow table with 1024 exact match flows and measured the delay to update a subset of 100 flows. Simultaneously, the measurement module polls the switch for full table statistics at a constant rate. The experiment uses a constant rate 10 Mbps packet probe to monitor the data path, and polls every 10 seconds for SNMP CPU values.

In this experiment, we controlled the probing rate for the flow statistics extraction mechanism, and we plotted the time necessary before the modified flows become active in the flow table. For each probing rate, we repeated the experiment 50 times, plotting the median, 10^{th} and 90^{th} percentile. In Figure 3.9 we can see that, for lower polling rates, implementations have a near-constant insertion delay comparable to the results

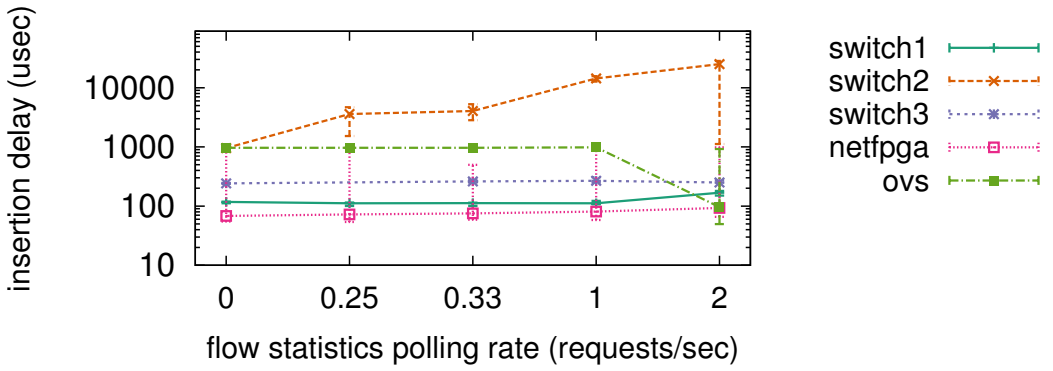


Figure 3.9: Delay when updating flow table while the controller polls for statistics.

of Section 3.4.3. For higher probing rates on the other hand, Switch1 and Switch3 do not differ much in their behaviour. In contrast, Switch2 exhibits a noteworthy increase in the insertion delay. This is explained by the CPU utilization increase incurred by the flow statistics polling (Figure 3.8(b)). Finally, Open vSwitch has a marginal decrease in the median insertion delay, and at the same time an increase in its variance. We believe that this behaviour is caused by interactions with the OS scheduling mechanism: the constant polling causes frequent interrupts for the user-space daemon of the switch, which leads to a batched handling of requests.

3.5 Network Control Macro-experimentation

OFLOPS and Cbench [Cbench, 2010] provide a sufficient toolbox for profiling the building blocks of OpenFlow, the switch and the controller. Nonetheless, in order to understand the impact of a control architecture, we require mechanisms that transform OpenFlow performance models into network-wide measurements under specific traffic patterns and network topologies. A popular approach to examine performance and correctness uses experimental setups which reproduce specific network-wide functionalities. Network experimentation uses two primary approaches: *emulation* and *simulation*.

Network emulation scales network system behaviour replication by replacing portions of network functionality with simpler models. Vahdat et al. [2002] present Mod-

elnet, an early network emulation framework attempt, which used highly optimized in-kernel resource control primitives to simulate large Internet-scale systems. Recent efforts employ lightweight virtualisation to scale large-scale network experiments [Gupta et al., 2011; Handigol et al., 2012b; Puljiz and Mikuc, 2006]. Network emulation approaches provide an effective solution to network experimentation, but the experimental fidelity is bound by the computational resources of the execution environment.

A notable approach to network emulation employs real testbeds to reproduce in full detail an experiment, and achieve maximum fidelity. Real testbeds incur significant resource and configuration overhead, which scale sub-linearly with respect to the size of the experiment. In an effort to improve resource scalability, the research community builds and maintains a series of shared testbeds, which multiplex experimental deployments in shared infrastructure using virtualisation [Ofelia, 2013; PlanetLab, 2007]. Shared testbeds fulfil the requirements for a wide range of experiments, but the resource sharing techniques limit measurement precision [Kim et al., 2011b], as they introduce noise which is not always detectable and removable.

Network simulation replaces network functionality with simplified models [Issariyakul, 2012; Varga and Hornig, 2008]. Simulation platforms set out to provide good resource scalability, but ultimately the accuracy and scalability is influenced by the assumptions of the network models and implementation. In addition, the programming abstractions in simulation platforms remain extensively incompatible with real system programming abstractions and, as a result, require a significant effort to transfer functionality between the two domains. For example, POSIX socket-based applications must modify their connection abstraction to match the API of the simulation platform, while forwarding plane traffic patterns may have to transform into stochastic models.

Network simulation and emulation follow two philosophically distinct approaches to scale experiments, each providing different trade-offs. In the section, we present SDNSIM[1], a network experimentation framework which bridges the two aforementioned approaches. The framework is written in OCaml, a high performance functional language, and extends the functionality of the Mirage[2] library OS. Experimenters can

[1]SDNSIM is under the GPL licence and can be downloaded from http://github.com/crotsos/sdnsim/

[2]*http://openmirage.org*

Appliance	*Binary size (MB)*
DNS	0.184
Web Server	0.172
OpenFlow switch	0.164
OpenFlow controller	0.168

Table 3.3: Sizes of Mirage application images. Configuration and data are compiled directly into the image.

describe network functionality over the Mirage OS systems API, and, during compilation, transform the experimental definition into a concrete experiment realisation. SDNSIM provides two experimentation options: *Simulation*, transforms the experiment definition into an ns-3 [Henderson et al., 2006] simulation, and *Emulation*, emulates the experiment definition using the Xen virtualisation platform [Barham et al., 2003].

3.6 Network Experimentation

This section presents the design of the SDNSIM platform. We present the Mirage library OS (Section 3.6.1), a core building block for the functionality of experimentation nodes, and elaborate on the functionality of SDNSIM and its integration with the ns-3 and Xen platform (Section 3.6.2).

3.6.1 Mirage Library OS

Mirage [Madhavapeddy et al., 2013] is a development framework for cloud services, supporting the transformation of applications in single purpose appliances, compile-time specialised into stand-alone Xen-compliant kernels. SDNSIM re-uses the Mirage abstraction and code to develop host functionality for two core reasons: the low memory footprint of Mirage applications; and its clean and simplified design. Mirage revisits the idea of library OS; functionality is separated into logical modules and linked to an appliance only if the code expresses an explicit dependency. As a result, Mirage by design results in compact OS images with very low memory requirements. Table 3.3 presents the compiled image size for a set of Mirage applications.

Mirage OS exposes a simple systems programming API, implemented by two core

modules: *Net* and *OS*. The OS module provides thread scheduling, device management and IO functionality, while the Net module provides a network stack supporting ICMP, IP, TCP, UDP and DHCP protocols. The interface of the core Mirage API is sufficiently generic to enable development of complex distributed services, while the OS architecture is sufficiently simple and portable over a wide range of platforms. Specifically, a Mirage application can compile into a Xen image, a Linux application or a NodeJS executable. Currently, the platform is being ported to the FreeBSD kernel and the BareMetal OS [Return Infinity, 2013], a high performance assembly-code OS. Furthermore, Table 3.4 presents the wide range of network protocols readily available in Mirage.

3.6.2 SDNSIM Architecture

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<topology module="Simple_tcp_test" backend="ns3-direct"
   slowdown="2.0" duration="30">
  <modules>
    <library>lwt</library>
    <library>lwt.syntax</library>
    <library>cstruct</library>
    <library>cstruct.syntax</library>
    <library>mirage</library>
    <library>mirage-net</library>
    <library>pttcp</library>
  </modules>
  <node name="host1" main="host_inner">
    <param>1</param>
  </node>
  <node name="host2" main="host_inner">
    <param>2</param>
  </node>
  <link src="host1" dst="host2" delay="10" rate="100"
    queue_size="100" pcap="false"/>
  <link_ext host="host1" dev="eth1" delay="10" rate="100"
    queue_size="100" pcap="false" ip="10.1.1.1">
</topology>
```

Listing 3.1: A sample SDNSIM configuration file interconnecting a server and a client

host

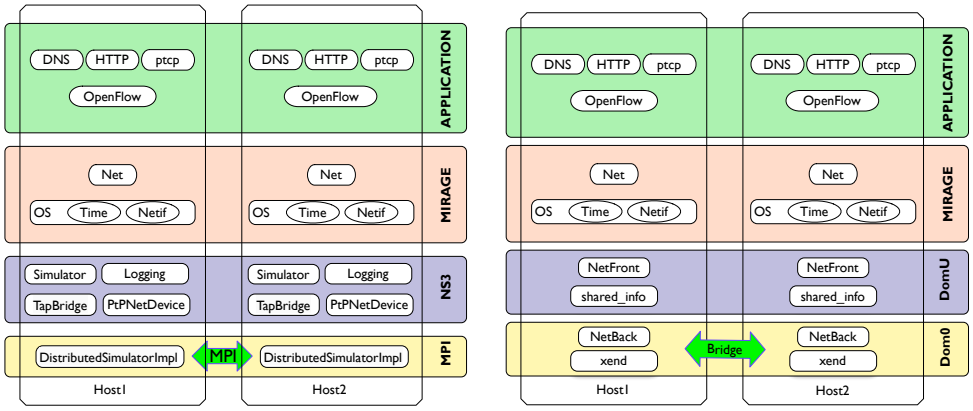
From the experimenter perspective, SDNSIM is an alternative build system for Mirage applications. Experimenters implement host functionality as Mirage applications and describe network topology using an XML-based configuration file. SDNSIM is responsible for compiling the Mirage applications and deploying the experiment, according to the configuration file.

Listing 3.1 presents an SDNSIM configuration file for a client (host1) - server (host2) architecture. A minimal experimental configuration consists of the name of the main OCaml module containing the host code (topology@module), the target experimentation environment (topology@backend) and the experiment duration (topology@duration). Optionally, the emulation backend supports a slowdown parameter (topology@slowdown) which controls a time dilation mechanism, presented in detail in Section 3.6.2.1. Hosts are defined using a `host` XML entity, containing a host name (node@name), its main function (node@main) and a variable number of children `param` entities (node/param) for host initialisation. Network link definitions use a `link` XML entity, containing the name of the interconnected hosts (link@src, link@dst), the device queue size (link@queue_size), the link propagation properties (link@rate, link@delay) and logging (link@pcap). Finally, experiments can import external network interfaces, allowing interaction with external applications. Links with external hosts are defined using a `link_ext` XML entity, which has the same parameters as the `link` entity, except from the `src` and `dst` attributes. These attributes are replaced by the host (link_ext@host) and dev (link_ext@dev) attributes, which map the network device of a host in the experiment to a TUN/TAP network device. In addition, the optional IP (link_ext@ip) attribute defines the address and netmask of the TUN/TAP device.

Figure 3.10 presents the host architecture in of SDNSIM nodes. Host functionality is separated in three layers. The top layer contains the application logic of the host. This layer is defined by the experimenter, and encodes the end-host network functionality. SDNSIM uses all application libraries of the Mirage platform, enumerated in Table 3.4, and thus provides high traffic realism. Furthermore, we extended the ocaml-openflow library to incorporate switch and controller performance models and implement the pttcp [Pratt, 2002] traffic generator functionality in OCaml, to support

Subsystem	*Implemented Protocols*
Network	Ethernet, ARP, DHCP, IPv4, ICMP, UDP, TCP, OpenFlow
Storage	Simple key-value, FAT-32, Append B-Tree, Memcache
Application	DNS, SSH, HTTP, XMPP, SMTP
Formats	JSON, XML, CSS, S-Expressions

Table 3.4: System facilities provided as Mirage libraries.



(a) ns-3 (b) Xen

Figure 3.10: SDNSIM host internal architecture: ns-3 simulation (Figure 3.10(a)) and Xen real-time emulation (Figure 3.10(b)).

stochastic traffic models. The middle layer of the host architecture contains the core Mirage libraries and provides OS functionality to the application layer. Finally, the lower layer of the host architecture, provides integration between the Mirage OS and the execution backend. Currently, SDNSIM has two experimentation backends: the *ns-3* simulation platform and the *Xen* virtualisation platform. These two execution environments are highly heterogeneous and employ different execution models. For the rest of this section, we present the integration details between Mirage and each experimentation backend. SNDSIM's main goal is network accuracy and the presentation focuses on the network and time subsystems.

3.6.2.1 Xen Backend

SDNSIM uses the Xen virtualisation platform to emulate experimental definitions. Hosts are represented as Mirage VMs, while network links are realised using virtual

interfaces (VIF) bridged in Dom0. SDNSIM replicate network link properties using the resource control primitive (e.g. throughput) of the XEN VIF implementation.

Experiment setup uses the Xen Management API (XAPI) [Xen Project, 2013], which provides an API to control programmatically the creation of VM instances and VIFs. Specifically, the execution pipeline for an emulated experiment is the following: host definitions are compiled in VM images, equivalent VM configuration are created on Xen and network links are translated into equivalent VIF and bridge setups. Once all configurations are completed, created VMs are booted and experimentation scenario is executed for a predefined duration.

As discussed in the previous section, a core limitation of experimental emulation is computation resource scalability. In order to address this problem, SDNSIM modifies the Mirage core libraries to support time dilation functionality. Specifically, when the resources of the execution host are insufficient, the experimenter can use a time dilation factor (TDF) to trade time for resource scalability. Time dilation slows down time progression synchronously across the emulated nodes, thus increasing the available resource in a time unit. For example, a TDF value of 2 means that 1 second of emulation time requires 2 seconds of real time to execute, doubling effectively the experimental perceived CPU and network resources. Similar approaches have been proposed in the domain of experimental emulation, in an effort to scale the complexity of large topologies. [Gupta et al., 2011] present a series of modifications in the Xen hypervisor, providing time virtualisation.

The clean design of the Mirage OS provides easy modification of the time subsystem. Mirage thread functionality relies on OCaml Lwt language extension, an event-driven asynchronous programming library. Lwt threads are cooperative and non preemptive, either executing code or idling waiting for an IO event or a synchronisation primitive (e.g. a semaphore) or a timer event. The main thread scheduler logic works as follows: Execute any resumable sleeping threads, calculate the time left until the next timer event and poll the IO event channel until that time expires. SDNSIM time virtualisation functionality requires only a multiplication of the time events and time sources with the TDF value. Guest hosts access time information through a shared page with the hypervisor containing a shared_info struct containing CPU time information. Mirage encapsulates access to the shared_info page through a single `time` function, which was modified with the TDF parameter. Similarly, we modify the time

event registration method to use the TDF parameter. Finally, time virtualisation is enforced on network resources by dividing link throughput values and multiplying link latency values with the TDF parameter.

3.6.2.2 ns-3 Backend

SDNSIM uses ns-3 [Henderson et al., 2006], a discrete-time packet-driven network simulator. The core of the system consists of a time simulation engine, while the code base provides multiple libraries modelling functionalities like network applications, routing protocols, data-link layer protocols and link propagation models. ns-3 is fundamentally an extensive refactor of the ns-2 code base, providing a stabler development environment.

The ns-3 backend has a significant difference from the Xen backend; the execution model is discrete and non-reentrant. ns-3 applications register their interest for specific time and network events, event handling is atomic and the progress of the simulation is centrally controlled by the simulation engine. In order to port Mirage to the ns-3 simulation engine, we transformed thread blocking calls into equivalent ns-3 events. More specifically, the Mirage OS clock abstraction is bridged with the ns-3 simulation clock and all sleep calls are scheduled as ns-3 time events. A thread is resumed from a sleep call when the simulator executes the respective time event. Network blocking calls are integrated with the network device abstraction of ns-3. The packet reading thread registers a packet handler on the network device, while the packet writing thread checks the device queue occupancy and blocks when the queue is full. Finally, in order to avoid scheduling deadlocks, the OS schedules *idle* time events to resume any yielded threads. Using these transformations we are able to provide a semantically accurate integration of Mirage with the ns-3 engine. Furthermore, network links between hosts are simulated using the *PointToPoint* link model. This model simulates a PPP link over a lossless medium, a valid approximation of the full duplex non-shared medium of current wired Ethernet technologies.

SDNSIM simulation backend use a distributed simulation engine for ns-3 which provides conservative event execution parallelisation. The simulation engine spawns a different process for each host of the simulation and an MPI-based synchronisation algorithm establishes conservative clock synchronisation and distributed event execu-

tion [Pelkey and Riley, 2011].

3.7 SDNSIM Evaluation

In this section, we evaluate SDNSIM performance and precision using small scale micro-benchmarks that target the OpenFlow protocol functionality and the ns-3 backend scalability. Madhavapeddy et al. [2013] present an exhaustive performance analysis of the Mirage platform, providing evidence on the data plane scalability of the Xen backend. In detail, this section presents a performance comparison of the Mirage OpenFlow controller with other controllers (Section 3.7.1) and the Mirage OpenFlow switch (Section 3.7.2) with the Open vSwitch software switch, and an evaluation the time scalability of the distributed ns-3 backend (Section 3.7.3).

3.7.1 Controller Emulation Scalability

We benchmark our controller library's performance scalability through a baseline comparison against two existing OpenFlow controllers, NOX and Maestro. NOX [Gude et al., 2008] is one of the first and most mature open-source OpenFlow controllers; in its original form it provides programmability through a set of C++ and Python modules. In our evaluation we compared it against both the master branch and the *destiny-fast* branch, an optimised version that sacrifices Python integration for better performance. Maestro [Cai et al., 2011] is a Java-based controller designed to provide service fairness between multiple OpenFlow control channels. We compared these controllers against our Xen-based Mirage OpenFlow controller application.

Our benchmark setup uses the *Cbench* [Cbench, 2010] measurement tool. Cbench emulates multiple switches which simultaneously generates `pkt_in` messages. The program measures the processing throughput of each controller. It provides two modes of operation, both measured in terms of `pkt_in` requests processed per second: *latency*, where only a single `pkt_in` message is allowed in flight from each switch; and *throughput*, where each emulated switch maintains a full 64 kB buffer of outgoing messages. The first measures the throughput of the controller when serving connected switches fairly, while the second measures absolute throughput when servicing requests from switches.

Controller	Throughput (kreq/sec)		Latency (kreq/sec)	
	avg	std. dev.	avg	std. dev.
NOX fast	122.6	44.8	27.4	1.4
NOX	13.6	1.2	26.9	5.6
Maestro	13.9	2.8	9.8	2.4
SDNSIM	98.5	4.4	24.5	0.0

Table 3.5: OpenFlow controller performance.

We emulated 16 switches concurrently connected to the controller, each serving 100 distinct MAC addresses. We ran our experiments on a 16-core AMD server running Debian Wheezy with 40 GB of RAM and with each controller configured to use a single thread of execution. We restricted our analysis to the single-threaded case as the Mirage execution environment, at time of testing, does not support multi-threading. For each controller we ran the experiment for 120 seconds and measured the per-second rate of successful interactions. Table 3.5 reports the average and standard deviation of requests serviced per second.

Unsurprisingly, due to mature, highly optimised code, *N*OX fast shows the highest performance for both experiments. However, the controller exhibits extreme short-term unfairness in the throughput test. *N*OX provides greater fairness in the throughput test, at the cost of significantly reduced performance. Maestro performs as well as NOX for throughput but significantly worse for latency, probably due to the overheads of the Java VM. Finally, Mirage throughput is somewhat reduced from NOX fast, but substantially better than both NOX and Maestro. In addition, the Mirage controller achieves the best product of performance and fairness among all tested controllers in the throughput test. Comparing latency, Mirage performs much better than Maestro, but suffers somewhat in comparison to NOX. From the comparison results, we conclude that the Mirage controller performance is comparable to the performance of existing controlling platforms and our emulation environment can reproduce controller deployments realistically. The SDNSIM time dilation mechanism can improve further the throughput of the system in the emulated time domain, providing sufficient computational headroom to emulate a wide range of existing controllers.

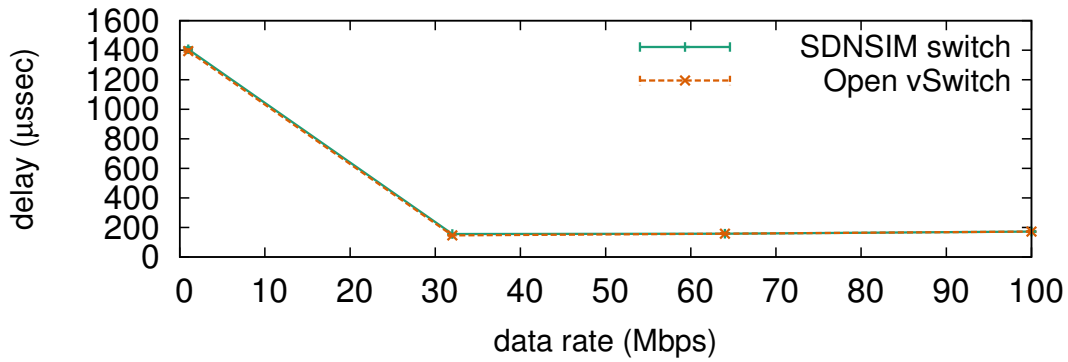


Figure 3.11: Min/max/median delay switching 100 byte packets when running the Mirage switch and Open vSwitch kernel module as DomU virtual machines.

3.7.2 Switch Emulation Scalability

We benchmarked our Mirage OpenFlow switch implementation through a baseline comparison with the Open vSwitch kernel implementation. We developed, using the OFLOPS framework, a simple forwarding test and measured the switching latency incurred by each implementation. For this experiment we used two virtual machines, one running the OFLOPS code, the other running the OpenFlow switch configured with three interfaces bridged separately in Dom0. One interface was used by the OpenFlow control channel, while the other two interfaces were used as switch ports. Using OFLOPS, we generated packets on one of the data channels and received traffic on the other, having bootstrapped appropriate the switch flow table. We ran the test for 30 seconds, a sufficient measurement period to detect statistically significant results. We used small packets (100 bytes)[1] and varied the data rate.

Figure 3.11 plots as error boxes the minimum, median and maximum of the median processing latency of ten test runs of the experiment. We can see that the Mirage switch's forwarding performance is very close to that of the Open vSwitch, even mirroring the high per-packet processing latency with a probe rate of 1 Mbps; we believe this is due to a performance artefact of the underlying Dom0 network stack. We omitted packet loss data, but can report that both implementations suffer similar levels of packet loss. From the comparison results of the Mirage OpenFlow switch, we can con-

[1]We used a packet size slightly larger that the minimum packet size because we appended in the payload packet generation information (e.g. packet ID, packet generation timestamps).

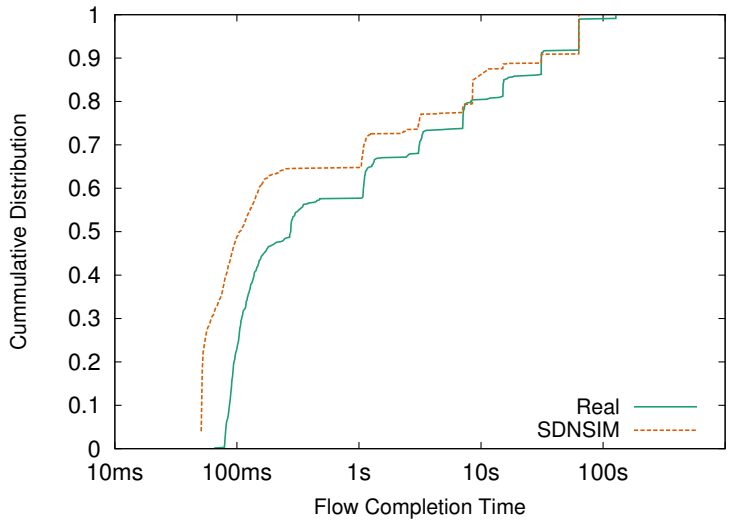


Figure 3.12: Flow completion time cumulative distribution for a real setup using Switch4 and an SDNSIM emulation using a model switch. The switch model can approximate with high accuracy the behaviour of the real system.

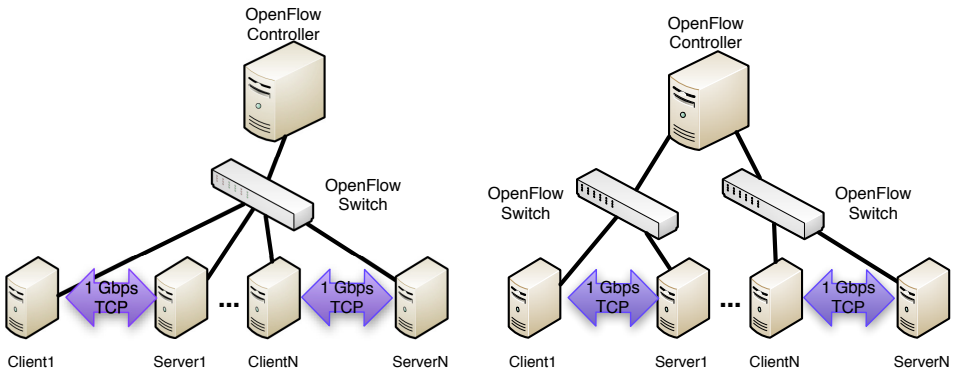
clude that our OpenFlow switch can emulate accurately software switch functionality. Nonetheless, the minimum latency of our switch emulation (approximately 100 ms) is 2 order of magnitude higher than a hardware switch (approximately 0.5-1 μs). This performance difference can be bridged easily using the time dilation functionality with TDF=100.

Furthermore, using the results from the evaluation of Switch4 (Section 3.4), we configured our switch model to evaluate the switch modelling functionality of SDNSIM. We set-up a simple topology with two hosts, a *data sink*, and a *data generation* host, interconnected through an OpenFlow switch. The data sink host generates flow requests to the data generation host following an exponential distribution with $\lambda = 0.02$ and a constant request size of 1MB. The measurement probe achieves an average 200 Mbps data rate. We ran our experiment for 120 seconds, ensuring that the resulting number of flows is below the flow table limits. For all experiments we used a learning switch control application.

Figure 3.12 presents the cumulative distribution of flow completion times for the real and model-based switch. The results highlight the impact of the switch control plane design on data plane performance. In the real experiment, 20% of flows have

higher completions time than 10 seconds. The predominant cause of this effect is the rate-limiting behaviour of the switch in `pkt_in` message transmissions. This results in TCP SYN and SYNACK packet drops and retransmissions during "bursty" periods. The completion time distribution exhibits a stepping behaviour, aligned with the back-off delay mechanism of TCP in the SYN_SENT/SYN_RCVD states[1]. Our model is able to capture the macroscopic behaviour of the switch, but exhibits a minor completion time overestimation. We attribute this behaviour to the high load that the reactive control scheme introduces to the communication channel between the co-processor of the switch and the ASIC, which skews the measured behaviour at run-time.

3.7.3 ns-3 Simulation Scalability



(a) centralised topology (b) distributed topology

Figure 3.13: Network topology to test the scalability of ns-3-based simulation. We simulated two topologies: a centralised topology (Figure 3.13(a)) and a distributed topology (Figure 3.13(b)).

We evaluated the SDNSIM simulation scalability using a simple network topology, depicted in Figure 3.13. The topology consisted of a number of switches and host pairs, connected through 1 GbE links. Each pair of hosts generated steady state TCP traffic at line rate. We used two variations of the topology: a *centralised topology* using a single switch to connect all hosts (Figure 3.13(a)), and a *distributed topology*,

[1]In our experiments we are using Linux hosts with kernel 3.11.0, which incorporates TCP Fast Open Cheng et al. [2013].

	Centralised topology			Distributed topology		
Number of hosts	4	8	12	4	8	12
Wall-clock delay (in min)	13	35	75	8	25	42
Slowdown factor	26	70	150	16	50	84

Table 3.6: Wall-clock delay to run 30 seconds of simulation time of the experiment in Figure 3.13, with a varying number of network hosts. SDNSIM simulation using the ns-3 backend scales linearly when the traffic is localised.

with pairs of hosts distributed between two switches (Figure 3.13(b)). Each switch is connected to a controller running a learning switch application.

We executed 30 seconds of simulation time on a 24-core AMD server with 128 GB of RAM and present in Table 3.6 the wall-clock execution time and the slowdown factor of each simulation. From the results we note that the real time to execute a simulation in SDNSIM depends on the number of network events; as we increase the number of host and, consequently, the number of packets, the execution time increases. Additionally, from the comparison between the centralised and distributed topology, we note that the performance of the simulation improves when network events are localized, as the simulation can be parallelised. In the distributed topology, network events are distributed between the two switches and execute independently, achieving near-linear scaling. The scaling properties of the SDNSIM simulation backend matches datacenter traffic experimentation, where application run independently and networks employ multipath topologies [Kandula et al., 2009].

3.8 Hierarchical Distributed Control

As we have discussed earlier in Section 2.4, the OpenFlow abstraction can be used both for reactive and proactive control. The two approaches represent the two extremes of the control spectrum: proactive control pre-computes forwarding policy and thus reduces control plane latency; while reactive control enables unprecedented fine levels of network control at the cost of latency and scalability. In this section we present a use case for SDNSIM, evaluating the impact of distributed reactive control in network performance.

The experimental design revisits the hierarchical design pattern on network control

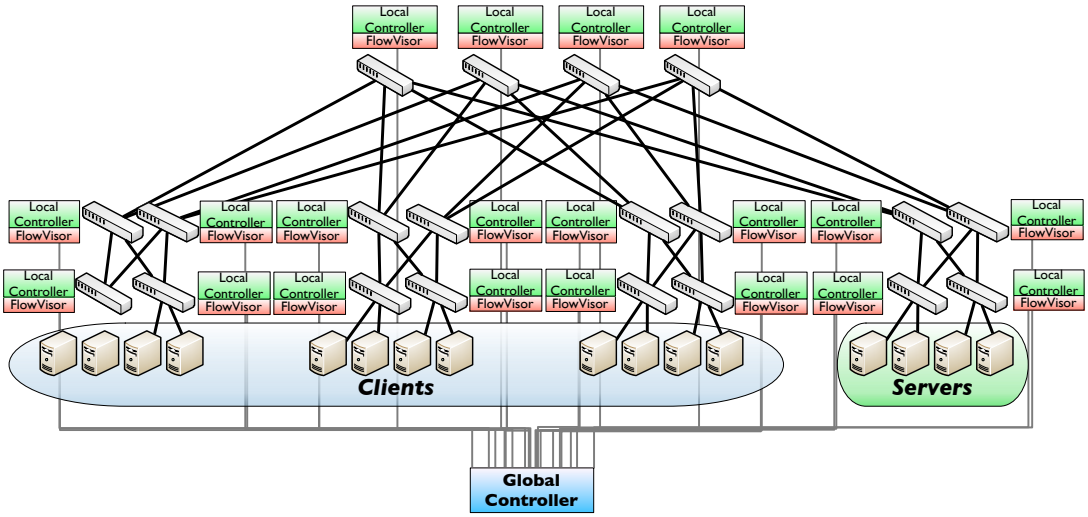


Figure 3.14: Hierarchical control experimental topology.

distribution. Specifically, a global controller connects to all network devices and maintains a global view of the network, while local controllers connect to a subset of the devices and exercise local forwarding policy in order to improve control responsiveness when necessary. A FlowVisor instance on each network device multiplex switch control channels between local and global controllers, and can dynamically delegate control between the two controllers. During periods with high control plane load, the architecture can switch control to the local controllers in order to improve latency and reduce the load on the central controller. Our experimental scenario simulates this control architecture and evaluates the impact of multi-level control on data plane performance.

The experiment, presented in Figure 3.14, uses a FatTree Clos topology with 4 pods and replicates the network behaviour of a database application. Specifically, the hosts in the first three pods act as clients, while the hosts of the fourth pod act as database servers. Each client selects uniformly at random a server and generates a 10 byte request, which is fulfilled by the server. The request size is selected uniformly at random from the values of 4 KB, 8 KB, 16 KB, 32 KB and 64 KB, while request arrivals follow a Poisson model. The control architecture of the network replicates the control logic presented by Al-Fares et al. [2008]. Every second the controller polls for flow-level statistics, calculates link loads, and redistributes flows between redun-

Parameter	value
packet_in processing delay	10 μsec
flow_mod insertion delay	120 μsec
link propagation delay	0.5 μsec
switch processing delay	10 μsec
link capacity	100 Mbps
Request arrival rate (per host)	62 retrievals/sec
Network device queue size	100 packets

Table 3.7: Hierarchical control simulation parameters.

dant network paths. In addition, the controller distributes new flow between redundant paths, with probabilities proportional to the path load ratio. The experiment tries to recreate with high fidelity the impact of SDN control in network functionality. In order to achieve this, we use a switch model based on the performance characteristics of Switch4. Nonetheless, the measured performance characteristics of Switch4 cannot accommodate the flow arrival rate exhibited by the employed traffic model. For example, the switch requires approximately 1 ms to install a flow in the flow table and the employed arrival rate of 250 flows per second would create a queue with utilization $\rho > 1$. As a result, we improve the performance of the employed model by an order of magnitude in order to support the experimental event rates. Our experimental definition incorporates additional experimental parameter regarding the controller performance characteristics and processing and propagation delays, presented in Table 3.8.

Using the previous experimental setup, we evaluated the impact of distributed and centralised control. Distributed control uses only local controllers and defines the forwarding policy using the traffic matrix of a single switch. In this setup we assume that a local controller has minimum latency to access the switch and minimizes the control channel latency and load. Global control uses only the global controller and setups end-to-end paths on a per-flow basis using the network-wide traffic matrix.

We evaluate the effect of each control scheme on data plane performance using the flow completion time metric, as proposed by relevant research [Dukkipati and McKeown, 2006]. Figure 3.15 presents the median, 10^{th} and 90^{th} quantiles of the completion time distribution for each flow size and for each control scheme. From the results we note two important observation. Firstly, the results point out the significant

non-linear increase in flow completion time between short (4 KB and 8 KB) and long flows (16 KB, 32 KB and 64 KB). This result is a direct consequence of the TCP slow start behaviour, as discussed by Dukkipati et al. [2010]. Mirage uses an initial TCP window size of 5840 bytes, sufficient to transmit all flow bytes within the slow start phase for short flows. Longer flows increase their window slower and thus increase the probability to experience transient queueing latencies. Secondly, the centralised policy improves resource utilisation across the network and thus reduces the effect of network queueing delays. Although the distributed approach exhibits lower control channel propagation delay, the suboptimal forwarding policy of the system increase the flow completion time, especially for long-lived TCP connections. More specifically, distributed control increases the median completion time by 2 ms in comparison to centralised control, while for 4 KB flows the delay increase is limited to 200 μs. Distributed control can reduce the impact of control channel latency during high flow arrival rates, incurs a significant non-negligible data plane performance degradation.

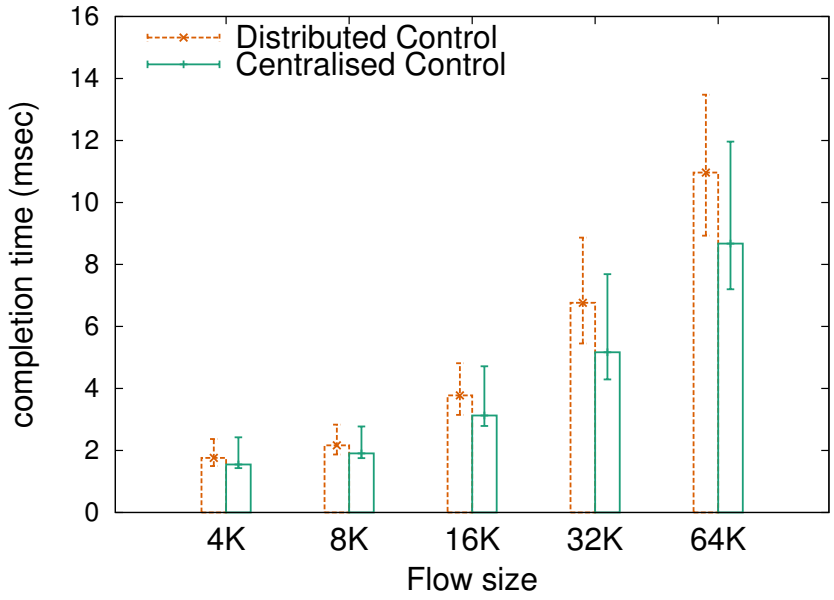


Figure 3.15: Comparison of TCP flow completion time for various flow sizes between a centralised and decentralised control architecture on a Fat-Tree topology.

3.9 Summary

This chapter discussed the scalability of the predominant SDN implementation, the OpenFlow protocol. We presented OFLOPS, a tool that tests the capabilities and performance of OpenFlow-enabled software and hardware switches. OFLOPS combines advanced hardware instrumentation, for accuracy and performance, and provides an extensible software framework. We used OFLOPS to evaluate six different OpenFlow switch implementations, in terms of OpenFlow protocol support as well as performance. In our performance evaluation, we benchmarked the packet processing, traffic interception and injection, flow table modification and traffic statistics export functionalities of the switches. We identified considerable variation among the tested OpenFlow implementations. We took advantage of the ability of OFLOPS for data plane measurements to quantify accurately how fast switches process and apply OpenFlow commands. For example, we found that the `barrier` reply message is not correctly implemented, making it difficult to predict when flow operations will be seen by the data plane. Finally, we found that the monitoring capabilities of existing hardware switches have limitations in their ability to sustain high rates of requests. Furthermore, at high rates, monitoring operations impact other OpenFlow commands.

In addition, in this chapter we presented the SDNSIM platform, an experimentation framework for large-scale high-precision experimentation with control plane architectures. SDNSIM used the Mirage abstraction and application libraries to enable experimenters to program the functionality of their network. An SDNSIM experiment definition can seamlessly translate at compile time into a simulation over the ns-3 simulation platform, or a high precision emulation over the Xen virtualisation framework. We conducted a number of micro-benchmarks and evaluated that the software provided emulation performance comparable to software OpenFlow controller and switching platforms, while SDNSIM simulation provided good scaling properties. Furthermore, we used SDNSIM to investigate the impact of hierarchical distributed control architectures on the performance of the data plane of the network. In the next chapter, we explore applications of the SDN paradigm to scale network management.

Chapter 4

Home network management scalability

In this chapter we explore applications of the SDN technology to scale network management in home networks. Drawing conclusions from existing social and user studies, we identify a series of user management requirements and we redesign the home network functionality and control abstraction addressing these requirements. We present a Strawman implementation of a network design which achieves simplicity and scalability of the control abstraction within the home environment. Additionally, we propose an extension of our design, which bridges the gap between the home network users performance requirements and the ISP resource allocation policy, providing a simple, scalable and user-friendly QoS mechanism.

In this chapter, motivated by the nature of the problems and the inherent opportunities of home networks (Section 4.1), we present a home router design (Section 4.2) and evaluate a number of proposed protocol modifications placing the homeowner in direct control of their network (Section 4.3). In addition, we present a simple QoS mechanism enabling user-ISP collaboration to improve resource utilisation in the last-mile bottleneck of the ISP (Section 4.4). Finally we conclude the results of our exploration (Section 4.5).

Note that throughout this Chapter we refer to the individual managing the home network as the homeowner without loss of generality; clearly any suitably authorised member of the household, owner or not, may be able to exercise control based on

specifics of the local context.

4.1 Motivations

In this section we elaborate on the nature of the problems and opportunities inherent to home networks. We motivate motivate our thesis by presenting the findings of relevant ethnographic and social studies (Section 4.1.1) and set the requirements for our system (Section 4.1.2).

4.1.1 Home Network: Use cases

Many empirical studies in recent years have explored the clear mismatch between current networking technology and the needs of the domestic setting, in both the UK [Crabtree et al., 2003; Rodden et al., 2004, 2007; Tolmie et al., 2007] and the US [Chetty et al., 2007; Grinter et al., 2005; Poole et al., 2008; Shehan and Edwards, 2007; Sung et al., 2007]. These studies present a weight of evidence that problems with home networking are not amenable to solution via a 'thin veneer' of user interface technology layered atop the existing architecture. Rather, they are *structural*, emerging from the mismatch between the stable 'end-to-end' nature of the Internet and the highly dynamic and evolving nature of domestic environments.

Home networks use the same protocols, architectures, and tools once developed for the Internet in the 1970s. Inherent to the Internet's 'end-to-end' architecture is the notion that the core is simple and stable, providing only a semantically neutral transport service. The core protocols were designed for a certain context of *use* (assuming relatively trustworthy endpoints), made assumptions about *users* (skilled network and systems administrators both using connected hosts and running the network core), and tried to accomplish a set of *goals* (e.g., scalability to millions of nodes) that simply do not apply in a home network.

In fact, the home network is quite different in nature to both core and enterprise networks. Existing studies [Poole et al., 2008; Shehan and Edwards, 2007; Tolmie et al., 2007] suggest domestic networks tend to be relatively small in size with between 5 and 20 devices connected at a time. The infrastructure is predominately cooperatively self-managed by residents who are seldom expert in networking management and, as

this is not a professional activity, rarely motivated to become expert. A wide range of devices connect to the home network, including desktop PCs, games consoles, and a variety of mobile devices ranging from smartphones to digital cameras. Not only do these devices vary in capability, they are often owned and controlled by different household members.

To illustrate the situation we are addressing, consider the following three example scenarios, drawn from situations that emerged from fieldwork reported in more detail elsewhere [Brundell et al., 2011; Chetty et al., 2010]:

> **Negotiating acceptable use**. William and Mary have a spare room which they let to a lodger, Roberto. They are not heavy network users and so, although they have a wireless network installed, they pay only for the lowest tier of service and they allow Roberto to make use of it. The lowest tier of service comes under an acceptable use policy that applies a monthly bandwidth cap. Since Roberto arrived from Chile they have exceeded their monthly cap on several occasions, causing them some inconvenience. They presume it is Roberto's network use causing this, but are unsure and do not want to cause offence by accusing him without evidence.
>
> **Welcome visitors, unwelcome laptops**. Steve visits his friends Mike and Elisabeth for the weekend and brings his laptop and smartphone. Mike has installed several wireless access points throughout his home and has secured the network using MAC address filtering in addition to WPA2. To access the network, Steve must not only enter the WPA2 passphrase, but must also obtain the MAC addresses of his devices for Mike to enter on each wireless access point. Steve apologizes for the trouble this would cause and, rather than be a problem to his hosts, suggests he reads his email at a local cafe.
>
> **Socially efficient network sharing**. Richard, the teenage son of Derek exhibits a great interest in online Music services. Derek works some times from home, using the Terminal Services provided by his employee. Tension is created between the household members, as Derek blames Richard downloading activity for his poor performing remote desktop application.

In such ways, simple domestic activities have deep implications for infrastructures that generate prohibitive technical overheads. In the first scenario, the problem is simply that the network's behaviour is opaque and difficult for normal users to inspect; in the second, the problems arise from the need to control access to the network and the technology details exposed by current mechanisms for doing so; in the third, the problem arises from the inability of the resource allocation algorithm to capture the social aspect of the home setting.

Home networks enable provision of a wide range of services, e.g., file stores, printers, shared Internet access, music distribution. The broad range of supported activities, often blending work and leisure, make network use very fluid. In turn, this makes it very hard to express explicitly *a priori* policies governing access control or resource management [Tolmie et al., 2007]. Indeed, fluidity of use is such that access control and policy may not even be consistent, as network management is contingent on the household's immediate needs and routines.

4.1.2 Home Networks: Revolution!

Current network functionality is spread across multiple layers that implement different abstractions, while multiple protocols are used to allow network hosts to communication over these layers. Each layer exposes a different set of control parameters and effective network management *must* exercise control on multiple layers. Ultimately, this control distribution architecture is not scalable for the average user. Simply creating a user interface layer for the existing network infrastructure will only reify existing problems. Rather, we need to investigate creation of new network architectures reflecting the socio-technical nature of the home by taking into account both human and technical considerations. Control of the network can be redefined, exposing only the required control and semantically appropriate abstraction, in order to scale controllability of the network.

To this end we exploit local characteristics of the home: devices are often collocated, are owned by family and friends who physically bring them into the home, and both devices and infrastructure are physically accessible. Essentially, the home's physical setting provides a significant source of heuristics we can understand, and offers a set of well understood practises that might be exploited in managing the infrastructure.

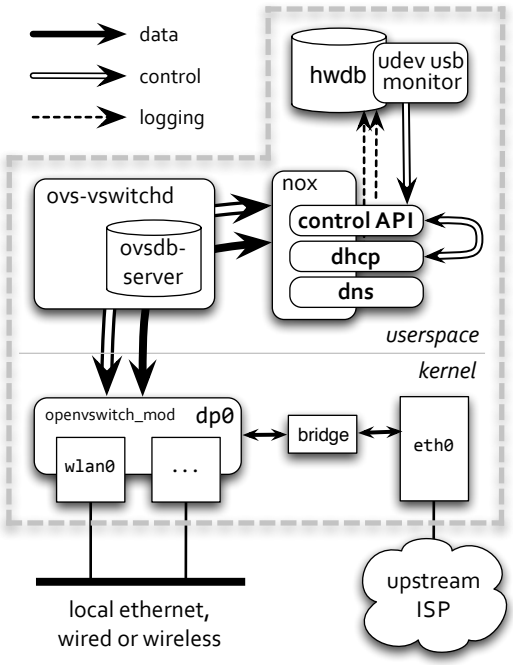


Figure 4.1: Home router architecture. Open vSwitch and NOX manage the wireless interface. Three NOX modules provide a web services control API, a DHCP server with custom address allocation and lease management, and a DNS interceptor, all logging to the Homework Database (*hwdb*) (§4.3).

We exploit human understandings of the local network and the home to guide management of the supporting infrastructure [Crabtree et al., 2003] by focusing on the home router not only as the boundary point in an edge network but as a physical device which can be exploited as a point of management for the domestic infrastructure. Within our router, we focus on flow management for three reasons:

- we do not require forwarding scalability to the same degree as the core network;
- doing so allows us to monitor traffic in a way that is more meaningful for users; and
- we can apply per-flow queueing mechanisms to control bandwidth consumption, commonly requested by users.

4.2 Reinventing the Home Router

Our home router is based on Linux 2.6 running on a micro-PC platform.[1] Wireless access point functionality is provided by the *hostapd* package. The software infrastructure on which we implement our home router, as shown in Figure 4.1, consists of the Open vSwitch OpenFlow implementation, a NOX controller exporting a web service interface to control custom modules that monitor and manage DHCP and DNS traffic, plus the Homework Database [Sventek et al., 2011] providing an integrated network monitoring facility. This gives us a setup very similar to a standard operator-provided home router where a single box acts as wireless access point, multiplexes a wired connection for upstream connectivity to the ISP, and may provide a small number of other wired interfaces.

We next describe the main software components upon which our router relies. Using this infrastructure, a number of novel user interfaces was developed, one of which is described briefly below; details of the others are available elsewhere [Mortier et al., 2011]. Note that a key aspect of our approach is to avoid requiring installation of additional software on client devices: doing so is infeasible in a home context where so many different types of device remain in use over extended periods of time.

4.2.1 OpenFlow, Open vSwitch & NOX

We provide OpenFlow support using Open vSwitch [Pettit et al., 2010] and employ the NOX [Gude et al., 2008] control platform, an event-driven programmable controller with C++ and Python module support, to develop our control logic. Our control logic, discussed in Section 4.3, is implemented in five distinct modules. The C++ module *hwdb* synchronizes router state with the hwdb home database, presented in Section 4.2.2, the C++ module *homework_dhcp* implements our custom DHCP server, the C++ module *homework_routing* implements the forwarding logic of the design, the C++ module *homework_dns* implements our DNS filtering functionality and the Python module *homework_rpc* exposes the control API through a Web service.

Our router provides flow-level control and management of traffic via a single OpenFlow datapath managing the wireless interface of the platform.[2] Control of the router

[1] An Atom 1.6GHz eeePC 1000H netbook with 2GB of RAM running Ubuntu 10.04.

[2] Without loss of generality, our home router has only a single wired interface so the only home-

Method	Function
permit/<eaddr>	Permit access by specified client
deny/<eaddr>	Deny access by specified client
status/[eaddr]	Retrieve currently permitted clients, or status of specified client
dhcp-status/	Retrieve current MAC–IP mappings
whitelist/<eaddr>	Accept associations from client
blacklist/<eaddr>	Deny association to client
blacklist-status/	Retrieve currently blacklisted clients
permit-dns/<e>/<d>	Permit access to domain d by client e
deny-dns/<e>/<d>	Deny access to domain d by client e

Table 4.1: Web service API; prefix all methods https://.../ws.v1/. $<X>$ and $[X]$ denote required and optional parameters.

is provided via a simple web service (Table 4.1). Traffic destined for the upstream connection is forwarded by the datapath for local processing via the kernel bridge, with Linux's *iptables* IP Masquerading rules providing standard NAT functionality.[1]

4.2.2 The Homework Database

In addition to Open vSwitch and NOX we make use of the Homework Database, *hwdb*, an active, ephemeral stream database [Sventek et al., 2011]. The ephemeral component consists of a fixed-size memory buffer into which arriving tuples (events) are stored and linked into tables. The memory buffer is treated in a circular fashion, storing the most recently received events inserted by applications measuring some aspect of the system. The primary ordering of events is time of occurrence.

The database is queried via a variant of CQL [Arasu et al., 2006] able to express both temporal and relational operations on data, allowing applications such as our user interfaces to periodically query the ephemeral component for either raw events or information derived from them. Applications need not be collocated on the router as *hwdb* provides a lightweight, UDP-based RPC system that supports one-outstanding-packet semantics for each connection, fragmentation and reassembly of large buffers,

facing interface is its wireless interface; other home-facing interfaces would also become part of the OpenFlow datapath.

[1] While NAT functionality could be implemented within NOX, it seemed neither interesting nor necessary to do so.

Figure 4.2: The *Guest Board* control panel, showing an HTC device requesting connectivity.

optimization of ACKs for rapid request/response exchanges, and maintains liveness for long-running exchanges. Monitoring applications request can execute temporal query on specific types of events. *hwdb* also provides notification functionality; applications may register interest in *future* behaviour patterns and receive notification when such patterns occur in the database. The work described in this paper makes use of three tables: *Flows*, accounting traffic to each 5-tuple flow; *Links*, monitoring link-layer performance; and *Leases*, recording mappings assigned via DHCP.

4.2.3 The Guest Board

This interface exploits people's everyday understanding of control panels in their homes, e.g., heating or alarm panels, to provide users with a central point of awareness and control for the network. This physical arrangement provides a focal point for inhabitants to view current network status and to manage the network. The interface provides a real time display of the current status of the network (Figure 4.2), showing devices in different zones based on the state of their connectivity. The display dynamically maps key network characteristics of devices to features of their corresponding labels. Mappings in the current display are:

- Wireless signal strength is mapped to device label transparency, so devices supplying weak signals fade into the background.
- Device bandwidth use is proportional to its label size, e.g., Tom's Laptop in

Figure 4.2 is currently the dominant bandwidth user.

- Wireless Ethernet retransmissions show as red highlights on the device's label, indicating devices currently experiencing wireless reliability problems.

Devices in range appear on the screen in real-time, initially in the leftmost panel indicating they are within range of the home router but not connected. The central panel in the control displays machines actively seeking to associate to the access point. This zone exploits the underlying strategy of placing people in the protocol, discussed in Section 4.3. When devices unknown to the network issue DHCP requests, the router's DHCP server informs the guest board and a corresponding label appears in this portion of the display. If a user wishes to give permission for the machine to join the network they drag the label to the right panel; to deny access, they drag the label to the left panel. The guest board provides both a central control point and, by drawing directly upon network information collected within our router, a network-centric view of the infrastructure.

4.3 Putting People in the Protocol

We use our home router to enable *ad hoc* control of network policy by non-expert users via interfaces such as the Guest Board (Figure 4.2). This sort of control mechanism is a natural fit to the local negotiation over network access and use that takes place in most home contexts. While we believe that this approach may be applicable to other protocols, e.g., NFS/SMB, LPD, in this section we demonstrate this approach via our implementation of a custom DHCP server and selective filters for wireless association and DNS that enable management of device connectivity on a per-device basis.

Specifically, we describe and evaluate how our router manages IP address allocation via DHCP, two protocol-specific (EAPOL and DNS) interventions it makes to provide finer-grained control over network use, and its forwarding path. We consider three primary axes: *heterogeneity* (does it still support a sufficiently rich mix of devices); *performance* (what is the impact on forwarding latency and throughput of our design and implementation decisions); and *scalability* (how many devices and flows can our router handle). In general we find that our home router has ample capacity

to support observed traffic mixes, and shows every indication of being able to scale beyond the home context to other situations, e.g., small offices, hotels.

4.3.1 Address Management

DHCP [Droms, 1997] is a protocol that enables automatic host network configuration. It is based on a four way broadcast handshake that allows hosts to discover and negotiate with a server their connectivity parameters. As part of our design we extend the functionality of the protocol to achieve two goals. First, we enable the homeowner to control which devices are permitted to connect to the home network by interjecting in the protocol exchange on a case-by-case basis. We achieve this by manipulating the lease expiry time, allocating only a short lease (30 seconds) until the homeowner has permitted the device to connect via a suitable user interface. The short leases ensure that clients will keep retrying until a decision is made; once a device is permitted to connect, we allocate a standard duration lease (1 hour).

Second, we ensure that all network traffic is visible to the home router and thus can be managed through the various user interfaces built against it. We do so by allocating each device to its own /30 IP subnet, forcing inter-device traffic to be IP routed via our home router. This requirement arises because wireless Ethernet is a broadcast medium so clients will ARP for destinations on the same IP subnet enabling direct communication at the link-layer. In such situations, the router becomes a link-layer device that simply schedules the medium and manages link-layer security – some wireless interfaces do not even make switched Ethernet frames available to the operating system. The result is that traffic between devices in the home, such as music distribution and file stores, becomes invisible to the home router. By allocating addresses from distinct subnets, all traffic between clients must be transmitted to the gateway address, ensuring all traffic remains visible to our home router. Our custom DHCP server allocates /30 subnet to each host from 10.2.*.*/16 with standard address allocation within the /30 (i.e., considering the host part of the subnet, 00 maps to the network, 11 maps to subnet broadcast, 01 maps to the gateway and 10 maps to the client's interface itself). Thus, each local device needs to route traffic to any other local device thought the router, making traffic visible in the IP layer.

We measured the performance of our DHCP implementation and found that, as

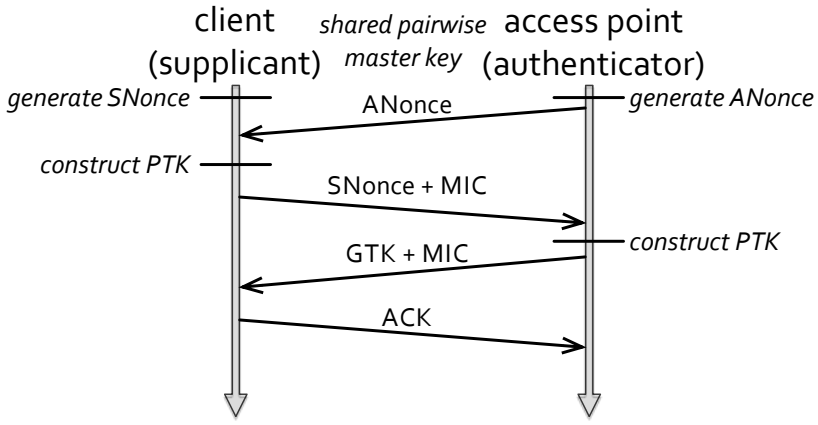


Figure 4.3: 802.11i handshake, part of the association process. Note that MIC (Message Integrity Code) is an alternate term for MAC, used in such contexts to avoid confusion with Media Access Control.

expected, per-request service latency scales linearly with the number of simultaneous requests. Testing in a fairly extreme scenario, simultaneous arrival of 10 people each with 10 devices, gives a median per-host service time of 0.7 second.

4.3.2 Per-Protocol Intervention

Our current platform intervenes in two specific protocols providing greater control over access to the wireless network itself, and to Internet services more generally.

Our home router supports wireless Ethernet security via 802.11i with EAP-WPA2, depicted in Figure 4.3, using *hostapd*. In EAP-WPA2 security mechanism, the client (*supplicant*) and our router (*authenticator*) negotiate two keys derived from the shared master key via a four-way handshake, through the EAPOL protocol. The *Pairwise Transient Key* (PTK) is used to secure and authenticate communication between the client and the router; the *Group Transient Key* (GTK) is used by the router to broadcast/multicast traffic to all associated clients, and by the clients to decrypt that traffic. All non-broadcast communication between clients must therefore pass via the router at the link-layer (for decryption with the source's PTK and re-encryption with the destination's PTK), although the IP routing layers are oblivious to this if the two clients are

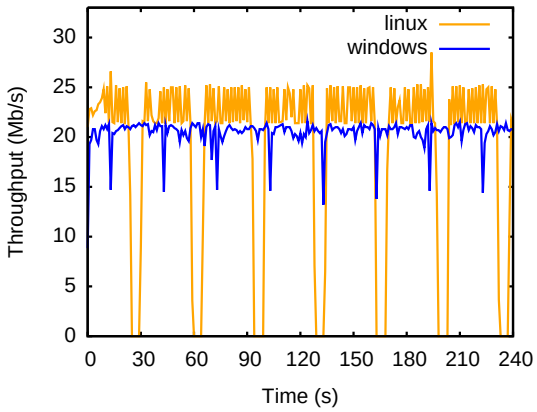


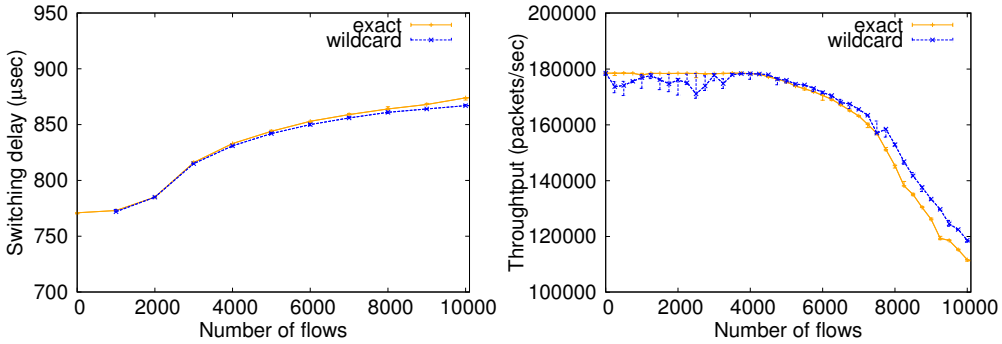
Figure 4.4: Affect on TCP throughput from 802.11i rekeying every 30 seconds for Linux 2.6.35 using a Broadcom card with the *athk9* module; and Windows 7 using a proprietary Intel driver and card.

on the same IP subnet.[1]

Periodically, a timeout event at the access point initiates rekeying of the PTK, visible to clients only as a momentary drop in performance rather than the interface itself going down. We use this to apply blacklisting of clients deemed malicious, such as a client that attempts to communicate directly (at the link-layer) with another client, i.e., attempting to avoid their traffic being visible to our home router. We wait until the rekeying process begins and then decline to install the appropriate rule to allow rekeying to complete for the client in question. This denies the client access even to link-layer connectivity, as they will simply revert to performing the four-way handshake required to obtain the PTK. This gives rise to a clear trade-off between security and performance: the shorter the rekeying interval, the quicker we can evict a malicious client but the greater the performance impact on compliant clients.

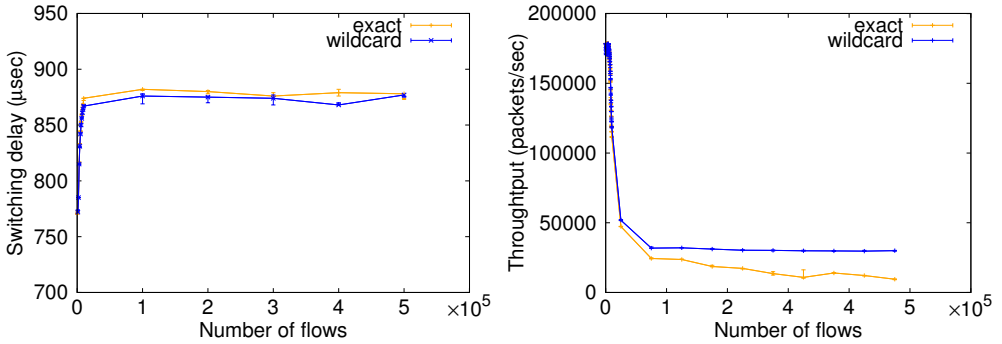
To quantify the impact of 802.11i rekeying, we observed throughput over several rekeying intervals. Figure 4.4 shows the transient impact to a steady-state TCP flow of setting the rekeying interval to 30 seconds: rekeying causes a periodic dip in throughput as the wireless Ethernet transparently buffers packets during rekeying be-

[1]The 802.11i specification defines a general procedure whereby two clients negotiate a key for mutual communication (*Station-to-station Transient Key*, STK). However, the only use of this procedure in the specification is in *Direct Link Setup* (DLS) used in supporting 802.11e, quality-of-service. This can easily be blocked by the access point, and in fact is not implemented in the *hostapd* code we use, so we do not consider it further.



(a) homework switching latency up to 10k concurrent flows

(b) homework packet throughput up to 10k concurrent flows



(c) homework switching latency up to 50k concurrent flows

(d) homework packet throughput up to 50k concurrent flows

Figure 4.5: Switching performance of Open vSwitch component of our home router showing increasing median per-packet latency (Figure 4.5(c), 4.5(a)) and decreasing median packet throughput (Figure 4.5(d), 4.5(b)) with the number of flows. The upper set of graphs (Figure 4.5(d),4.5(c)) extends the x-axis from 10,000 to 500,000.

fore transmitting them as if nothing had happened. This shows the trade-off between performance and responsiveness of this approach: to be highly responsive in detection of misbehaving clients imposes a small performance degradation. As a compromise, when a device is blacklisted, all of its traffic and subsequent rekeying exchanges are blocked. Thus the misbehaving device is prevented from sending or directly receiving any traffic before rekeying takes place, The device will be able to receive only broadcast traffic in the interim due, to the use of the GTK for such frames, until the AP initiate the negotiation of a new key. This allows us to pick a relatively long rekey interval (5 minutes) while still being able to respond quickly to misbehaving devices.

We also intercept DNS to give fine-grained control over access to Internet services and websites. DNS requests are intercepted and dropped if the requesting device is not permitted to access that domain. Any traffic the router encounters that is not already permitted by an explicit OpenFlow flow entry has a reverse lookup performed on its destination address. If the resulting name is from a domain that the source device is not permitted to access, then a rule will be installed to drop related traffic. Performance is quite acceptable, as indicated by latency results in Figure 4.5: the extra latency overhead introduced by our router is negligible compared to the inherent latency of a lookup to a remote name server.

4.3.3 Forwarding

Our router consists of a single Open vSwitch that manages interface *wlan0*. Open vSwitch is initialised with a set of flows that push DHCP/BOOTP and IGMP traffic to the controller for processing. Open vSwitch by default will also forward to the controller traffic not matched by any other installed flow, which is handled as follows:

Non-IP traffic. The controller acts as a proxy ARP server, responding to ARP requests from clients. Misbehaving devices are blacklisted via a rule that drops their EAPOL [Aboba et al., 2004] traffic thus preventing session keys negotiation. Finally, other non-IP non-broadcast traffic has source and destination MAC addresses verified to ensure both are currently permitted. If so, the packet is forwarded up the stack if destined for the router, or to the destination otherwise. In either case, a suitable OpenFlow rule with a 30 second idle timeout is also installed to shortcut future matching traffic.

Unicast IP traffic. First, a unicast packet is dropped if it does not pass all the following tests:

- its source MAC address is permitted;
- its source IP address is in 10.2.x.y/16; and
- its source IP address matches that allocated by DHCP. For valid traffic destined to the Internet, a flow is inserted that forwards packets upstream via the bridge and IP masquerading.

Unicast IP traffic that passes but is destined outside the home network has a rule installed to forward it upstream via the bridge and IP masquerading. For traffic that is to remain within the home network a flow is installed to route traffic as an IP router, i.e. rewriting source and destination MAC addresses appropriately. All these rules are installed with 30 seconds idle timeouts, ensuring that they are garbage collected if the flow goes idle for over 30 seconds.

Broadcast and multicast IP traffic. Due to our address allocation policy, broadcast and multicast IP traffic requires special attention. Clients send such traffic with the Ethernet broadcast bit[1] set, normally causing the hardware to encrypt with the GTK rather than the PTK so all associated devices can receive and decrypt those frames directly. In our case, if the destination IP address is all-hosts broadcast, i.e., 255.255.255.255, the receiver will process the packet as normal. Similarly, if the destination IP address is an IP multicast address, i.e., drawn from 224.*.*.*/4, any host subscribed to that multicast group will receive and process the packet as normal. Finally, for local subnet broadcast the router will rebroadcast the packet, rewriting the destination IP address to 255.255.255.255. This action is required because the network stack of the hosts filters broadcast packets from different IP subnets.

To assess switching performance, we examine both latency and packet throughput as we increase the number of flows, N, from 1–500,000. In this measurement we employ a PC, functioning as a data generator and receiver, connected directly over a 1 Gbps full-duplex Ethernet link to our home route configured to forward all received packets back to the incoming port. Each test runs for two minutes, a sufficient period to observe the network performance behaviour, generating packets at line rate from a

[1] I.e., the most significant bit of the destination address

single source to N destinations each in its own 10.2.*.*/30 subnet, creating equally N unique network flows. As these are stress tests we use large packets (1500 bytes) for the latency tests and minimal packets (70 bytes)[1] for the throughput tests, selecting destinations at random on a per-packet basis. Results are presented as the median of 5 independent runs with error bars giving the min and max values.

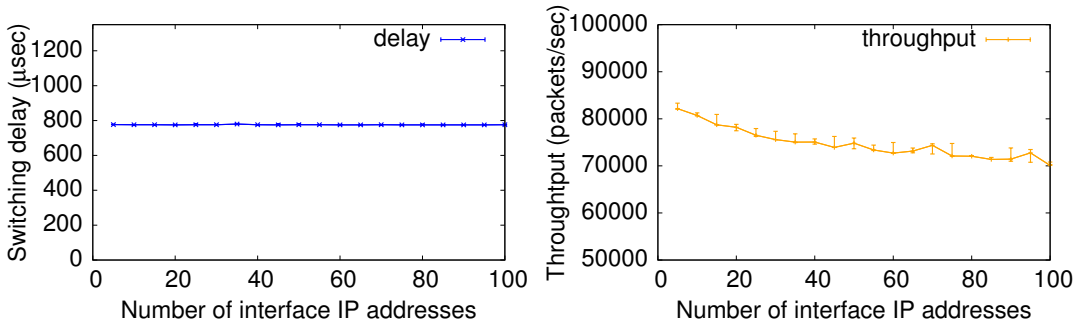
Figure 4.5 shows median per-packet switching delay and per-flow packet throughput using either exact-match rules or a single wildcard rule per host. Performance is quite acceptable with a maximum switching delay of 560 μs and minimum throughput of 40000 packets/second (Figure 4.5(c),4.5(a)); initial deployment data suggests a working maximum of 3000 installed flows which would give around 160000 packets/second throughput (small packets) and 500 μs switching delay (large packets) (Figure 4.5(a),4.5(b)). Using a similar topology we evaluate the performance of multihoming in Linux hosts. Figure 4.6 shows that the Linux networking stack is quite capable of handling the unusual address allocation pattern resulting from the allocation of each wireless-connected device to a distinct subnet which requires the router's wireless interface to support an IP address per connected device. Increasing the number of assigned IP address has no impact on the processing latency and minimizes marginally the maximum packet processing rate. This performance behaviour can also be explained by the trie data structure, used by the Linux kernel to implement longest prefix match, which scale lookup complexity logarithmically with respect to the number of IP addresses.

4.3.4 Discussion

Our evaluation shows that Open vSwitch can handle orders of magnitude more rules than required by any reasonable home deployment. Nonetheless, to protect against possible denial-of-service attacks on the flow tables, whether accidental or malicious, our home router monitors the number of per-flow rules introduced for each host. If this exceeds a threshold then the host has its per-flow rules replaced with a single per-host rule, while the router simultaneously invokes user interface callback to inform the homeowner of the device's odd behaviour.

The final aspect to our evaluation is compatibility: given that our router exercises

[1]The 30 bytes extra overhead is due to *pktgen* [Olsson, 2005], the traffic generation tool used.



(a) Packet throughput

(b) Packet latency

Figure 4.6: Switching performance of Linux network stack under our address allocation policy. Throughput (Figure 4.6(a)) shows a small linear decrease while switching delay (Figure 4.6(b)) remains approximately constant as the number of addresses allocated to the interface increases.

protocols in somewhat unorthodox ways, how compatible is it with standard devices and other protocols? We consider compatibility along three separate dimensions: range of existing client devices; deployed protocols that rely on broadcast/multicast behaviours; and support for IPv6.

Devices Although we exercise DHCP, DNS and EAPOL in unorthodox ways to control network access, behaviour follows the standards once a device is permitted access. To verify that our home router is indeed suitable for use in the home, we tested against a range of commercial wireless devices running a selection of operating systems.

Table 4.2 shows the observed behaviour of a number of common home-networked devices: in short, all devices operated as expected once permitted access. DNS interception was not explicitly tested since, as an inherently unreliable protocol, all networking stacks must handle the case that a lookup fails anyway. Most devices behaved acceptably when denied access via DHCP or EAPOL, although some user interface improvements could be made if the device were aware of the registration process. The social context of the home network means no problem was serious: in practice the user requesting access would be able to interact with the homeowner, enabling social negotiation to override any user interface confusion.

Device	Denied	Blacklisted
Android 2.x	Reports pages unavailable due to DNS.	Retries several times before backing off to the 3g data network.
iTouch/iPhone	Reports server not responding after delay based on configured DNS resolver timeout.	Requests new wireless password after 1–2 minutes.
OSX 10.6	Reports page not found based on configured DNS resolver timeout.	Requests new wireless password after 1–2 minutes.
Microsoft Windows XP	Silently fails due to DNS failure.	Silently disconnects from network after 4–5 minutes.
Microsoft Windows 7	Warns of partial connectivity.	Silently disconnects from network after 4–5 minutes.
Logitech Squeezebox	Reports unable to connect; allows server selection once permitted.	Flashes connection icon every minute as it attempts and fails to reconnect.
Nintendo Wii	Reports unable to reach server during "test" phase of connection.	Reports a network problem within 30 seconds.
Nokia Symbian OS	Reports "can't access gateway" on web access.	Reports disconnected on first web access.

Table 4.2: Observed interactions between devices and our home router when attempting to access the network.

Broadcast protocols A widely deployed set of protocols relying on broadcast and multicast behaviours are those for 'zero conf' functionality. The most popular are Apple's *Bonjour* protocol; *Avahi*, a Linux variant of Bonjour; Microsoft's *SSDP* protocol, now adopted by the UPnP forum; and Microsoft's *NetBIOS*.

Bonjour and Avahi both rely on periodic transmission of multicast DNS replies advertising device capabilities via TXT records. SSDP is similar, but built around multicast HTTP requests and responses. We tested Bonjour specifically by setting up a Linux server using a Bonjour-enabled daemon to share files. We observed no problems with any clients discovering and accessing the server, so we conclude that Bonjour, Avahi and SSDP would all function as expected.

NetBIOS is somewhat different, using periodic network broadcasts to disseminate hosts' capabilities. In doing so we observed a known deficiency of NetBIOS: it cannot propagate information for a given workgroup between different subnets.[1] However, installing a WINS server on the router mitigates this problem.

In general, it may seem that our address allocation policy introduces link-layer overhead by forcing all packets to be transmitted twice in sending them via the router.

[1] http://technet.microsoft.com/en-gb/library/bb726989.aspx

However this is not the case: due to use of 802.11i, unicast IP traffic between two local hosts must *already* be sent via the access point. As the source encrypts its frames with its PTK, the access point must decrypt and re-encrypt these frames with the destination's PTK in order that the destination can receive them. Multicast and all-hosts broadcast IP traffic is sent using the GTK, so can be received directly by all local hosts. Only directed broadcast IP traffic incurs overhead which though is a small proportion of the total traffic; data from a limited initial deployment (about one month in two homes) suggests that broadcast and multicast traffic combined accounts for less that 0.1% (packets and bytes) in both homes.

IPv6 support IPv6 support is once more receiving attention due to exhaustion of the IPv4 address space. Although our current implementation does not support IPv6 due to limitations in the current Open vSwitch and NOX releases,[1] we briefly discuss how IPv6 would be supported on our platform. While these limitations prevent a full working implementation in our platform, we have verified that behaviour of both DHCPv6 and the required ICMPv6 messages was as expected, so we do not believe there are any inherent problems in the approaches we describe below.

Addition of IPv6 support affects the network layer only, requiring consideration of routing, translation between network and link layers, and address allocation. Deployment of IPv6 has minimal impact on routing, limited to the need to support 128 bit addresses and removal, in many cases, of the need to perform NAT [2]. Similarly, supporting translation to lower layer addresses equates to supporting ICMPv6 Neighbour Solicitation messages which perform equivalent function to ARP.

Address allocation is slightly more complex but still straightforward. IPv6 provides two address allocation mechanisms: *stateless* and *stateful*. The first allows a host to negotiate directly with the router using ICMPv6 Router Solicitation and Advertisement packets to obtain network details, IP netmask and MAC address. Unfortunately this process requires that the router advertises a 64 bit netmask, of which current plans allocate only one per household, with the result that all hosts would end up on the

[1] OpenFlow provides support in its 1.4 release of the protocol; NOX currently has no support for IPv6 and OpenFlow versions beyond 1.0; and Open vSwitch only supports IPv6 as a vendor extension of the OpenFlow protocol.

[2] Some operators may still prefer to use NAT as part of a legacy of address management and operations.

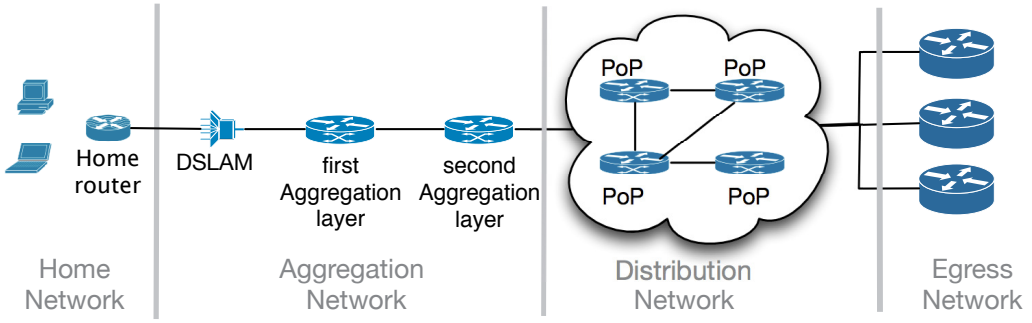


Figure 4.7: ISP network architecture is logically divided in 3 zones: The *Aggregation Network*, the *Distribution Network* and the *Egress Network*.

same subnet. The second builds on DHCPv6 where addresses are allocated from a central entity and may have arbitrary prefix length. This would enable our router to function in much the same manner as currently, although it would need to support the ICMPv6 Router Advertisement message to support router discoverability by hosts.

In order to test the functionality, we set up a simple hard-coded version of our design. Specifically, we allocate a public IPv6 /64 prefix to our home router. The router uses the ISC DHCPv6 server implementation, configured to allocate addresses from a /120 subnet. Additionally, on the router we run the RADVD daemon to reply appropriately to ICMPv6 messages. Using this network setup, we tested MacOSX, Windows and Linux IPv6 network stacks and verified correct network configuration and Internet connectivity.

4.4 Putting people in the Resource Allocation Policy

As we have discussed in Section 1.1.1, providing strong performance guarantees in large production networks using commodity technologies exhibits poor scalability. There are two primary approaches followed by the network community, to scale resource allocation policies: *network resource over-provision*; and *application-aware traffic engineering*. Resource over-provision increases available resources thus removing performance bottlenecks. Application-aware network engineering extends the network policy to manage efficiently application traffic, e.g. VoIP traffic is serviced by a separate high-priority VLAN. Nonetheless, the applicability of these approaches

in broadband networks is limited. On the one hand, resource over-provision faces a non-scalable deployment cost on the edges of the network [1], while, on the other hand, application-aware network engineering raises legal issues for Internet providers [Hahn and Wallsten, 2006]. In this section, we propose an architectural extension in the last-mile of the ISP, enabling homeowners to define per-application resource requirements. The architecture improves the ISP resource allocation policy, as well as, the user experience. For the rest of this section, we discuss the architecture of current broadband networks and their inherent limitations in resource allocation (Section 4.4.1), we present the architecture of our system (Section 4.4.2), and we evaluate its impact on specific applications classes (Section 4.4.3).

4.4.1 ISP Resource Allocation

In order to motivate the reader on the benefits of our architecture, we firstly present the architecture of current broadband networks and identify their performance bottleneck. Figure 4.7 presents a generic model for the network architecture for a residential broadband ISP. Although, exact architecture details remain undisclosed, there is a high level design pattern, organising the ISP network in three domains: the *aggregation network*; the *distribution network*; and the *egress network*. The aggregation network contains the digital subscriber line access multiplexer (DSLAM) which translates DSL packets to Ethernet frames, the Broadband Remote Access Server (BRAS), an authenticating and policy enforcing network service, and a series of aggregation switches which multiplex traffic between the BRAS and the distribution network. Traffic from the aggregation network routes through the distribution network towards either the egress network or towards a local aggregation switch. The distribution network consists of a small number of large routers interconnected using high capacity links. ISPs typically employ network tunnelling technologies, like MPLS, in the distribution network to minimize forwarding state. The egress network of an ISP consists of large routers, hosted in Internet Exchange Points (IXP), which connect the ISP with peering ASes.

A number of research papers have identified a significant bottleneck in the last-mile of the network [Akella et al., 2003; Dischinger et al., 2007]: the link between

[1] Optical link installation costs vary between \$50-\$200 per foot, while fiber installation in municipal areas has an annual cost of \$1.91 per foot [Carrier Ethernet News, 2011]. Such upgrades preclude a significant upgrade cost for equipment.

the DSLAM and the aggregation network. This bottleneck can be explained by the high oversubscription ratio of such links, which commonly varies between 10:1 and 50:1 [Broadband Canada Program, 2013], depending on population density of an area and market dynamics [Leach, 2013].

Our system aims to bridge a fundamental communication gap between the ISP and homeowners. Users perceive network traffic as an ensemble of flows belonging to a specific application and priority class within the home social context, e.g. Skype VoIP traffic has a higher priority than web browsing traffic and parents' teleworking traffic is more important than kids' entertainment traffic. ISPs view household traffic as a packet aggregate with a specific IP address, on which resource allocation mechanisms apply monthly or daily traffic caps [British Telecoms, 2013; Virgin Media, 2014]. Network caps rate-limit heavy-hitting households and provide long-term fairness between users. Nonetheless, this approach degrades severely the short-term performance for latency-bound applications, such as remote desktop and VoIP. Effectively, there is an incentive for ISPs and homeowners to collaborate and control more efficiently the traffic resources within the user cap. Homeowners can annotate high priority flows within their traffic aggregate and thus improve their experience. ISPs can increase the user-friendliness of their network policy and offload some of the resource allocation complexity to end-users.

Our design virtualises network control and resources in the last-mile and delegates control to a homeowner-managed OpenFlow controller. The household controller uses the OpenFlow protocol to assign traffic flows to appropriate priority queues. While the proposed solution cannot always guarantee end-to-end resource allocation, it provides a sufficient mechanism to handle in a user-friendly manner edge-network congestion, and scale the configuration complexity.

4.4.2 Architecture Design

Congestion in the last-mile requires resource control both in the home network and within the ISP network. Home traffic is asymmetric and the network bottleneck lies beyond the control of the homeowner, thus requiring control on both traffic directions. Our architecture consists of three subsystems: a data-collection daemon on the end-systems of the home network, a FlowVisor instance in the ISP network translating user

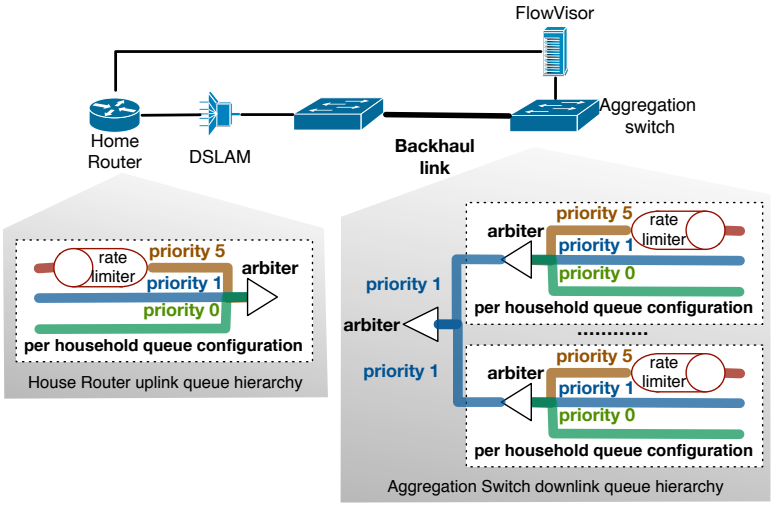


Figure 4.8: User-driven network resource allocation architecture. A virtual control slice of the downlink between the backhaul network and the DSLAM is exposed to each homeowner. The switch uses three queues per-household: A *low latency, high priority* queue, a *medium priority* queue and a *default* queue.

flow prioritisation into forwarding policy, and a policy enforcing daemon on the home network router.

Data collection on End-systems In order to map network flows to applications, we developed a light cross-platform daemon recording the content of the end-host connection table. The daemon collects mappings between flow tuples and applications names. Information is collected from the connection table of the OS and stored in the hwdb database. The daemon accesses the connection table entries using the *netfilter-conntrack* library [netfilter.org project, 2010] in Linux systems, the *Windows Filtering Platform* [Microsoft, 2012] in Windows systems and the *sysctl* for Darwin/MacOSX system. In order to match each network tuple with an application, we use the `lsof` command in Unix-like systems and the `netstat -p` command in Windows.

ISP policy translation Figure 4.8 presents the network architecture of our system. The resource mechanism exercises control on the home router and on the first layer of aggregation switches. The ISP exposes an OpenFlow control abstraction which virtualises switch resources on the aggregation switches using FlowVisor [Sherwood

et al., 2010b], while each home router runs an OpenFlow controller, controlling the home Internet traffic. The FlowVisor is configured to provide a clean separation between household control domains in order to mitigate control or eavesdropping attacks between households. `pkt_in` messages are propagated only to the respective home controller based on IP addresses, and the home controller can only view and insert flows containing public IP of the home network. In the household, the resource control mechanism augments the control logic, presented in the previous section. In order to minimize forwarding state in the ISP network, the control abstraction on the aggregation switches is used to control only the incoming Internet traffic, while the outgoing traffic can be effectively controlled on the local router.

At the aggregation switch and the home router, we setup three traffic shaping queues for each household. A *low latency, high priority* (LLHP) queue, a *medium priority* (MP) queue and a default queue. The LLHP queue has the highest priority between the household queues and rate limits traffic to the minimum guaranteed bandwidth per-household (e.g. for 1 Gbps uplink with a sharing ratio of 1:100, the LLHP queue will limit traffic to 10 Mbps), providing strong low-bandwidth and low-latency guarantees. The MP queue doesn't enforce any rate limit, but it has a priority which is between the LLHP and the default queue. MP queue can be used by latency-tolerant high-priority applications. Finally, the default queue is used to handle all other traffic types. The household queue hierarchies are multiplexed towards the output port using a Round-Robin scheduler with equal priority for each household. Such large scale queue hierarchies are currently supported by service provider network equipment (e.g. Cisco ASR 9000 routers provide 500000 queues per line card [Cisco, 2012b]) and ISPs use this functionality to implement their capping policy. By default, all network flows are forwarded to the default queue and the home OpenFlow controller at runtime assigns flows dynamically to higher priority queues. The enhanced control of our network architecture provides some interesting building blocks to enable novel economical models for residential broadband customers. For example, users can enhance network performance by purchasing additional flow table entries or by increasing their minimum guaranteed bandwidth on the edge.

Because our system design provides extensive network control to end-users, we fortify the design with a set of mechanisms to reduce the possibility of network functionality compromise. Firstly, fair allocation of flow table entries to each household,

ensures fair utilisation of the forwarding resources in the aggregate switch. Secondly, the FlowVisor instance rate limits OpenFlow control channel interactions per household to mitigate DoS attacks. Finally, the FlowVisor instance discards flows that may create loops in the network, e.g. flow rules that forwards packet to the incoming port.

Home OpenFlow controller In our design, the home router is the rendez-vous point between the policies of the two entities, responsible to store traffic statistics, record the user application priority, and at run-time map application flows to the appropriate queue. We use three tables in HWDB to store relevant information: `application tuple`; `application timeseries`; and `application priority`; tables. The `application tuple` table stores mappings between applications and flow tuples populated with data from the data-collection daemon. The `application timeserries` table stores the per-application network utilization, populated with router data using `flow_stats` messages. Finally, the `application priority` table stores the user's application prioritisation policy.

We expose resource control of our system to home members through a web interface, presented in Figure 4.9. The interface presents the aggregate and timeserries resource consumption for each application, using the data of the `application timeseries` table, and enables control of application prioritisation. Additionally, the interface notifies the user when the policy over-utilises the LLHP queue. Specifically, the packet loss counter from the OpenFlow `queue_stats` message is used to detect high packet loss incidents. Through this interface, we address issues raised by Chetty et al. [2010].

During operation, the proposed design extends the forwarding logic described earlier. Specifically, each valid network flow destined to a non-local host is mapped to an application, through the `application tuple` table, and to a priority queue, through the `application priority` table. The controller sends a `flow_mod` message with an `enqueue` action to assign the flow to the appropriate queue. In addition, if the application priority is not the default, then the controller will also send a `flow_mod` packet to the FlowVisor instance to setup the incoming direction of the flow.

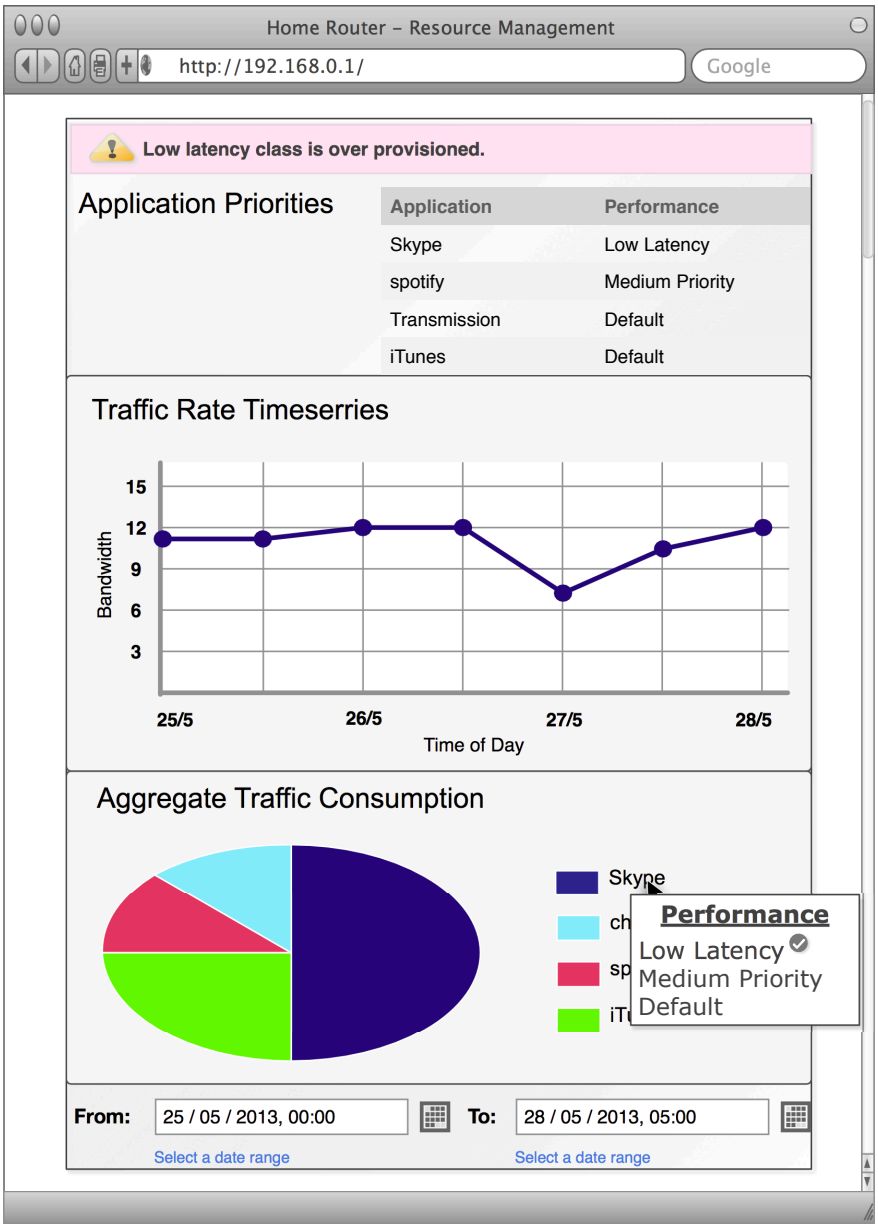


Figure 4.9: QoS user interface. The user is able to view the network utilisation per application, as well as express application prioritisation.

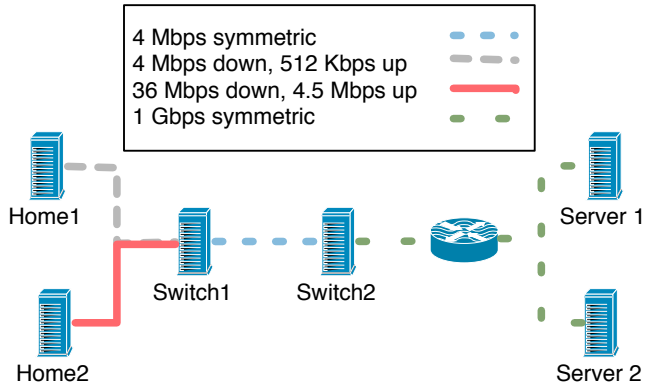


Figure 4.10: QoS mechanism evaluation topology.

Web traffic		**Gaming traffic**	
Parameter	**Distribution**	**Parameter**	**Distribution**
objects per page	Pareto(2.43, 3.00)		
inter-request delay	Pareto(1.50, 0.75)	inter-packet gap	Lognormal(79.34, 0.25)
inter-object delay	Weinbull(1.46, 0.38)	packet size	Constant(0.05)
object size	Lognormal(1.31, 9.35)		

Table 4.3: Model parameters for HTTP [Barford and Crovella, 1998] and gaming [Lang et al., 2004] traffic generation.

4.4.3 Evaluation

In this section, we evaluate the impact of our architecture on performance-sensitive applications. Figure 4.10 presents the topology we employ for our evaluation. *Home1* and *Home2* emulate household Internet traffic, *Server1* and *Server2* emulate Internet services, *Switch1* emulates the last-mile of the ISP network and *Switch2* emulates the Internet path between the aggregation layer of the ISP and the Internet services. In detail, Home1 emulates a single household running a resource critical application towards Server1, while Home2 emulates an ensemble of nine households generating bursty HTTP requests towards a web service on Server2, using the web model in Table 4.3. Switch1 provides asymmetric Internet links with 4 Mbps downlink capacity and 512 Kbps uplink capacity to each household and connects them to the ISP distribution network through a 4 Mbps symmetric link, thus providing a downstream subscription ratio $1 : 10$. Switch2 propagates traffic between Switch1, Server1 and Server2, adding 10 ms of one-way latency to each direction. Each node in the topology runs

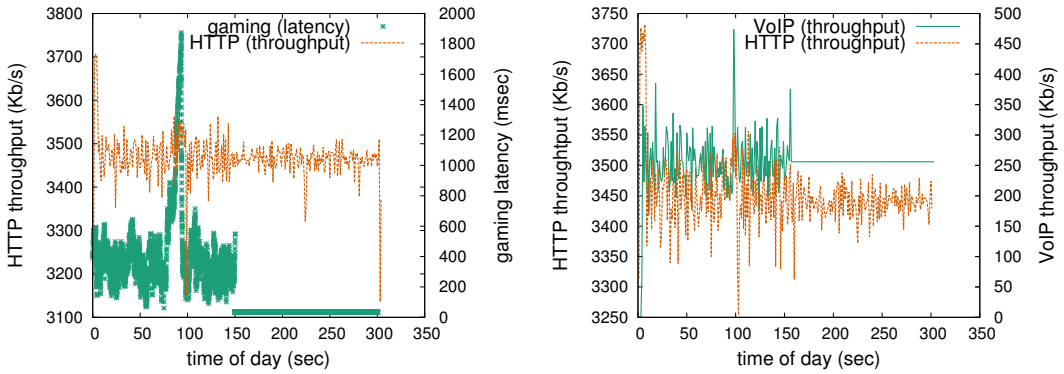
on a different host equipped with a quad core Intel CPU (Core 2 Quad Q6600), 4 GB RAM and a quad-port 1 GbE Intel card. Switch1 and Switch2 use Open vSwitch to implement fast packet forwarding functionality.

Using the aforementioned topology, we evaluate the impact of our architecture on two popular applications, VoIP and Gaming. Without loss of generality, the selected applications provide a representative set in terms of latency and throughput requirements. We emulate VoIP traffic, using a constant rate 250 Kbps TCP flow [1], while for gaming traffic we use a UDP probe based on the model presented in Table 4.3. We conduct the measurement for 300 seconds, initializing the network without any traffic prioritisation, and we enable our architecture after 150 seconds of execution, assigning the HTTP application on the default queue and the VoIP and Gaming applications on the LLHP queue. Figures 4.11(a) and 4.11(b) present the results of the experiment for the gaming and VoIP applications respectively. More specifically, we present the per-packet latency for gaming and the instantaneous application-perceived throughput for VoIP, along with the instantaneous aggregate throughput of all HTTP flows. For the gaming application, we observe that packets face significant latencies (200-500 ms), primarily due to queueing delays when they coexisting in a congested link with HTTP flows and the network is not using any short-term resource control. The latency is effectively minimized when we enable our architecture without significant performance degradation for HTTP traffic. Similarly, VoIP traffic experiences significant throughput variations (200-350 Kbps) which are eliminated when the flow is assigned to the LLHP queue. The results verify that the proposed short-term resource control mechanism can significantly improve user experience and effectively improve network performance for applications with strict resource requirements, while introducing minimal impact on the performance of less critical flows.

4.5 Summary

This chapter has drawn upon previous user studies to reflect on the distinct nature of home networks and the implications for domestic network infrastructures. Three particular user needs that arose from home network user studies were for richer visibility

[1] https://support.skype.com/en/faq/FA1417/how-much-bandwidth-does-skype-need



(a) Gaming (b) VoIP

Figure 4.11: Latency and throughput performance of the QoS architecture. The QoS mechanism is enabled after 150 seconds of execution, reducing significantly packet latency and providing accurate throughput bounds.

into and greater control over the home wireless network, as well as the home as part of the everyday management of the home by inhabitants. We considered how to exploit the nature of the home network in order to shape how it is presented and opened to user control and ultimately to provide scalable management through a novel control plane architecture.

We used the Open vSwitch and NOX platforms to provide flow-based management of the home network. As part of this flow-based management, we exploited the social conventions in the home to manage introduction of devices to the network, and their subsequent access to each other and Internet hosted services. This required modification of three standard protocols, DHCP, EAPOL and DNS, albeit in their behaviour but *not* their wire formats, due to the need to retain compatibility with legacy deployed stacks. In addition, in our exploration we presented a simple mechanism enabling home user to guide the ISP resource allocation mechanism in order to fulfil application requirements. Our exploration, on one hand, provides strong evidence that appropriate design of the control plane architecture can transform and scale the management effort, while on the other hand, it suggests that existing presumptions could usefully be re-examined to see if they still apply in a network context. In the next chapter we explore application of SDN technologies in naming and connectivity scalability.

Chapter 5

Secure and Scalable Internet-scale Connectivity

In this chapter we explore applications of the SDN technology in connectivity and naming scalability. The work is motivated by the user requirement to create network federations between its devices. We call this functionality a *Personal Cloud*. Currently, Personal Clouds use third-party Cloud services as information caches, to bridge the inter-device communication requirements, due to various connectivity and naming limitations of the Internet architecture as discussed in Section 1.1. Nonetheless, as users offload personal information to third-party services, they become subject to privacy threats and performance degradation.

We argue that end-devices require an adaptable network control architecture, in order to overcome connectivity limitations and control their data and resources. The network community provides an abundance of free and open mechanisms, which can be reused dynamically to provide inter-device connectivity. We introduce Signpost [Rotsos et al., 2013], a network control plane for end-devices, capable of delivering continuous and secure connectivity between user devices. Signpost uses existing network testing approaches to detect network environment condition and automate the configuration of existing software packages in order to establish inter-device connectivity.

In order to understand the feasibility and performance of the proposed architecture, we present a strawman implementation of the core control logic and its integration with a number of network connection and notification services. Currently, Signpost supports

SSH [Levin and Even, 2005], OpenVPN [OpenVPN, 2014], TOR [Dingledine and Mathewson, 2006], NAT-punch and Privoxy [Privoxy, 2013] mechanisms, while it can also propagate Multicast-DNS notifications across devices.

For the rest of this chapter, we present a preliminary categorisation of Personal Cloud platforms and, motivated by some key observations, we define the key design requirements for our platform (Section 5.1). Furthermore, we present the Signpost abstraction (Section 5.2) and architecture (Section 5.3), followed by an elaborate description of a strawman Signpost implementation and an evaluation of its functionality (Section 5.4). Finally, we conclude with our observations (Section 5.5).

5.1 Personal Clouds

The digital revolution of our era has mirrored a large part of our life in the digital domain. For example, human labour is commonly stored in PDF and word processing files, while social interactions are partially captured in digital photo collections. Effectively, our physical presence has an increasing digital counterpart, accessible through computing devices. Nonetheless, the increase in the adoption of personal computers introduces challenges in the management of our digital footprint. Complimentary to personal computers, users own and use tablets, smartphones and other purpose-built computing units (e.g. home entertainment systems and gaming consoles), while the IoT paradigm [A. et al., 2010] will extend further the available interaction points. From a user perspective, each device fulfils a specific role in managing digital resources, thus requiring access only to a subset of the user's digital presence, but these roles are fluid. For example, a smartphone is primarily a communication device, but end-users may use it as a media player. In the latter case, the smartphone and the home entertainment system provide similar functionality to the user and require access to the same subset of his digital presence. As a result, the transition from a single PC to a multi-device paradigm requires global and homogeneous access and control mechanisms of digital information and resources. We define this abstraction as the user's *Personal Cloud*.

This section presents existing Personal Cloud mechanisms, discusses their functional limitations (Section 5.1.1), and defines the core Signpost goals (Section 5.1.2).

5.1.1 Approaches

Personal Cloud functionality is readily available in a number of applications, exposing a wide range of abstractions. This section provides a simple taxonomy, based on their architecture.

Decentralised Personal Cloud The decentralised Personal Cloud application class contains standalone connectivity mechanisms, managed by end-users. Computer networks by design provide the middleware to enable information and resource sharing between computers. As a result, the network community provides an extensive collection of applications and protocols that support Personal Cloud functionality, such as remote desktop access (e.g. Remote Frame Buffer protocol [Richardson and Levine, 2011]), file sharing (e.g. NFS [Nowicki, 1989]; SAMBA file service [Leach and Naik, 1997]), remote login (e.g. SSH [Ylonen and Lonvick, 2006]), remote printing (e.g. IPP [Hastings et al., 2000]). Such systems employ a client-server architecture and extend the computer abstraction to include inter-device connectivity. A user can set-up and manage sharing services on his personal devices and enhance his computing experience. Furthermore, the setup complexity of some services has motivated the development of automation mechanisms like Multicast-DNS, UPnP and ZeroConf, which allow a user to seamlessly browse and access available services.

Decentralised Personal Cloud applications provide extensive control over personal data and strong privacy guarantees. Decentralised approaches are a popular mechanism for sharing data within the local network, like the home network, Nonetheless, Internet-scale deployment of such mechanism faces significant scalability problems, primarily due to the following three limitations:

- *Connectivity*: Decentralised Personal Cloud effectiveness is defined by Internet connectivity limitations. The current Internet architecture does not guarantee global and homogeneous service accessibility, as we have discussed in Section 1.1, limiting the applicability of client-server models. End-users commonly connect to the Internet through edge networks optimised for outgoing connectivity, while ubiquitous middleboxes limit service accessibility. For example, NATed networks restrict by default incoming Internet connectivity, while mobile and enterprise networks enforce network policies, using firewalls, which

limit device connectivity for security reasons.

- *Authentication*: In the context of security, there are two primary authentication models: out-of-band communication; and pre-deployed information [Touch et al., 2008]. The majority of decentralised applications use pre-deployed authentication mechanisms, based either on password or public key schemes. Although the approach is effective, it exhibits limited flexibility in supporting management automation and responsiveness to user credential compromises. Out-of-band authentication mechanisms effectively address the aforementioned problems, but existing applications often lack support, while enabling such mechanisms requires significant management effort.

- *Usability*: Decentralised Personal Cloud applications introduce significant usability challenges to inexperienced users. Configuration interfaces of relevant software vary significantly, while in some cases effective configuration requires a deep understanding of the system (e.g. the SSH client in Linux has 26 command line parameters and 70 configuration options). In addition, due to the previously discussed connectivity limitations, users must resort to a "trial and error" approach in order to evaluate the effectiveness of an application configuration in a specific environment, and potentially use multiple mechanisms to achieve connectivity, like tunnelling software. Effectively, the configuration of decentralised Personal Cloud applications is complex, time-consuming, and highly disengaging for inexperienced users, as discussed in Section 4.1.
  As an example, we will describe the required steps to access files from a remote device using the SSH service, following the decentralised approach. Initially, the user must configure the SSH service on the server host, both in terms of access control and service functionality, and any firewall and NAT device deployed in the local network. Additionally, the user must dynamically configure a naming service if the public IP of the server is dynamic, e.g. DynDNS[1]. SSH connectivity is subject to the ability of the user in the remote network to use the SSH protocol on the preconfigured port. In cases where this is not possible, the user has to try a different connection-establishing mechanism.

[1] http://dyn.com/dns/

Centralised Personal Cloud A popular alternative to decentralised Personal Cloud services uses third-party services to share information and resources between devices. In this class of applications we consider applications like the Google service suite, the Dropbox file sharing service, the LogMeIn Remote Desktop service, and other similar services. These applications provide a simple, intuitive and ubiquitous mechanism for sharing data and resources between a user's devices, or even with the user's social circle. Centralised Personal Cloud applications rely on resilient, high availability and responsive services. Although this approach provides an effective solution, some of its properties are ambivalent and demotivate user engagement.

- *Identity*: Cloud applications provide an effective control framework to disseminate information between devices and users. Each user has an online identity, verifiable by the service provider. Users can define information access policies using these identities and guarantee secure delivery. Nonetheless, such identities are susceptible to privacy attacks. Krishnamurthy and Wills [2009] present a privacy attack using online identities from online social networks. Specifically, the service provider exposed identification breadcrumbs to advertising services, thus allowing user monitoring. Facebook has openly verified the existence of such services [Perez, 2007].

- *Performance*: The elasticity of cloud computing infrastructures, commonly used to host centralised Personal Cloud applications, provides high performance scalability. Nonetheless, such applications commonly underutilize the wealth of computational and network resources available in edge networks and devices. Two devices connected to the same subnet will experience bloated RTT when they connect using a centralised service, while the cost and latencies of cloud storage are orders of magnitude higher in comparison to local storage. Wittie et al. [2010] report that Internet path latency and packet losses extensively affect the 95th percentille of cloud service network performance, while Dean and Barroso [2013] present a wide range of parameters which influence user-perceived performance, spanning from the CPU scheduler to hardware design in the cloud infrastructure.

- *Cost*: The majority of relevant services provide weak user service level agreements (SLA) and minimum guarantees to users and their data. As a result, if the

service is compromised and sensitive information is leaked, the SLA minimize the legal obligations of the provider towards affected users [McCullagh, 2011]. In addition, recent allegations [Greenwald and MacAskill, 2013] claim intentional data access rights being granted to unauthorized entities by large cloud and service providers. Although a significant number of such services provide cost-free access, the cost of personal information leaks can be significant and unobserved by the user [Liu et al., 2011].

- *Availability*: Cloud services run on well connected datacenters, managed by highly-skilled engineers, ensuring performance, functionality and security. However, centralised architectures have by design a single point of failure. For example, two devices, belonging to different users and connected to the same local network, cannot exchange a file over Dropbox if the Dropbox service is inaccessible[1].

5.1.2 Design Goals

In the previous section, we present the capabilities and limitations of existing Personal Cloud approaches. Decentralized Personal Cloud applications lack user-friendliness and their functionality is subject to the network policy. Centralised Personal Cloud applications introduce privacy vulnerabilities and are a performance bottleneck. The security and performance trade-offs are a direct consequence of the scalability limitations in current Internet architecture.

We believe that an efficient and secure Personal Cloud framework should employ a hybrid approach to scale connectivity, security and usability. User information must remain on user devices and the system must provide user-controlled privacy. The wealth of available decentralized Personal Cloud applications cannot support global connectivity, and thus functionality, and require cloud support for the establishment of an inter-device control channel. Nonetheless, cloud mediation is not always required in order to provide inter-device connectivity, as the host can evaluate the network policy and negotiate the use of a connection-establishing software, like tunnelling software.

[1] Dropbox provides local network synchronization functionality between devices. However it is only used for performance reasons and requires service connectivity https://www.dropbox.com/help/137/en

Signpost sets the following high-level architectural goals:

- *Naming*: A sufficient architecture requires a high availability naming mechanism translating global names into accessible network end-points.
- *Connectivity*: A Personal Cloud architecture must provide connectivity, and thus functionality, in all network environments. The system should be able to recover when exogenous factors disrupt connectivity.
- *Control*: A Personal Cloud system should expose a user-friendly access control abstraction. Access policies should operate on the level of devices or users and rely on strong authentication and encryption mechanisms. Additionally, the control should be sufficiently dynamic to respond to security changes, e.g. propagate trust revocation for a compromised device. The control abstraction should also incorporate diverse security aspects and allow the user to control the trade-offs between performance and security.
- *Backward Compatibility*: The framework should provide support to existing decentralised applications without any code modification.
- *Usability*: The system should expose a simple control abstraction to end-users in order to orchestrate the connection establishment process. All low-level network interactions should be managed by the system and remain hidden by the end-user. Ultimately, the abstraction of the system should be compatible with the existing functionality of decentralised Personal Cloud.

5.2 Finding Your Way With Signpost

Signpost is a decentralised naming and connectivity framework with user-controlled security. The system provides an Internet-wide overlay network between the devices of a user, enabling device federation. Signpost uses a cloud-based inter-device control channel to coordinate end-to-end network path establishment between devices by using existing connection-establishing software and protocols. In order to ensure functionality under any circumstance, the Signpost control channel is integrated with the Internet naming service, a high availability and resilient service.

At the core of the Signpost architecture, we define a generic testing and configuration model which automates connection establishment using a wide range of relevant software. Section 5.3.1 presents in detail the proposed model and Section 5.4.1 presents the integration with a representative set of connection-establishing software. Signpost offers backwards-compatibility with existing applications on the network layer. The applications can use Signpost paths by routing traffic to Signpost addresses, while the control API for inter-device connection establishment is integrated with the *gethostbyname()* OS method. Each device is exposed to the users as a domain name, and a name lookup for a device triggers the establishment of a Signpost path.

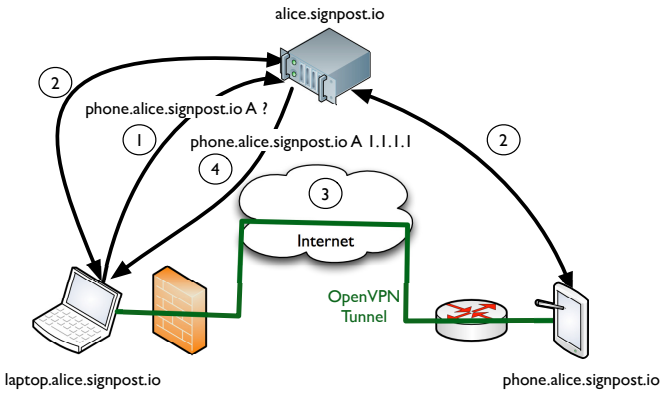


Figure 5.1: An example use-case of the Signpost abstraction. Alice establishes a direct communication channel between her smartphone and her home PC over the Internet.

In order to introduce the reader to the Signpost abstraction, Figure 5.1 depicts a simple use case example. In this scenario, Alice, while at work, wishes to access files from her laptop, situated behind a home router, through her smartphone. In order to express her interest in connecting to her laptop, Alice performs a name lookup for the domain name `laptop.bob.signpost.io` to her local Signpost resolver (step ①). The name request propagates through the public DNS infrastructure to her Signpost cloud service, which we call the *Signpost controller*. A verified name request to the Signpost controller initiates the connection establishment logic of the system, which triggers the two devices to try multiple connection establishment mechanisms and setup an end-to-end path (step ②). Once an initial path is available (step ③), the controller replies to the initial name request with a local Signpost-specific IP address. Traffic to this address is routed by the device network stack to the established inter-

device network path (step ④). In parallel, Signpost will continue to evaluate different connection mechanisms, aiming to discover paths with higher performance, and monitors path availability, thus recovering automatically from disconnections.

5.3 Signpost Architecture

Figure 5.2 presents the architecture of Signpost from three different viewpoints. In the lower section of the figure, we present the design of Signpost and its integration with existing applications. Signpost logic is contained in a single executable, and requires the guest OS to expose a flow control abstraction, like OpenFlow, and to redirect DNS queries to the embedded DNS resolver. The software consists of three subsystems: The *Connection Engine* (Section 5.3.3), responsible for setting up and managing *Network Tactics* (Section 5.3.1) for end-to-end path establishment, the local DNS resolver, and the OpenFlow-based *Signpost router* (Section 5.3.2), which enhances the normal OS routing functionality with Signpost network control logic. The middle section of Figure 5.2, presents the inter-device connectivity architecture of Signpost. The system sets up an Internet-wide inter-device control channel, enabling capability and parameter negotiation between devices. The architecture uses a special Signpost node, which we call *Signpost Controller*, to run on a well-connected host and bridge the device control channel across the Internet, effectively orchestrating connection negotiation and establishment. The upper section of Figure 5.2 presents the naming organisation of the Signpost architecture. The system reuses the naming abstraction of the DNS service. Each device has a global domain name, while the domain hierarchy and name aliasing expresses the control relationship between devices and users. DNS naming logic is augmented by Signpost, introducing an *effectful name resolution* functionality for Signpost-enabled domains; a name resolution expresses user interest in connecting to another device, and triggers the test and setup of a network path (Section 5.3.4). In the rest of this section we present in detail the architecture of Signpost, using a bottom-up narrative.

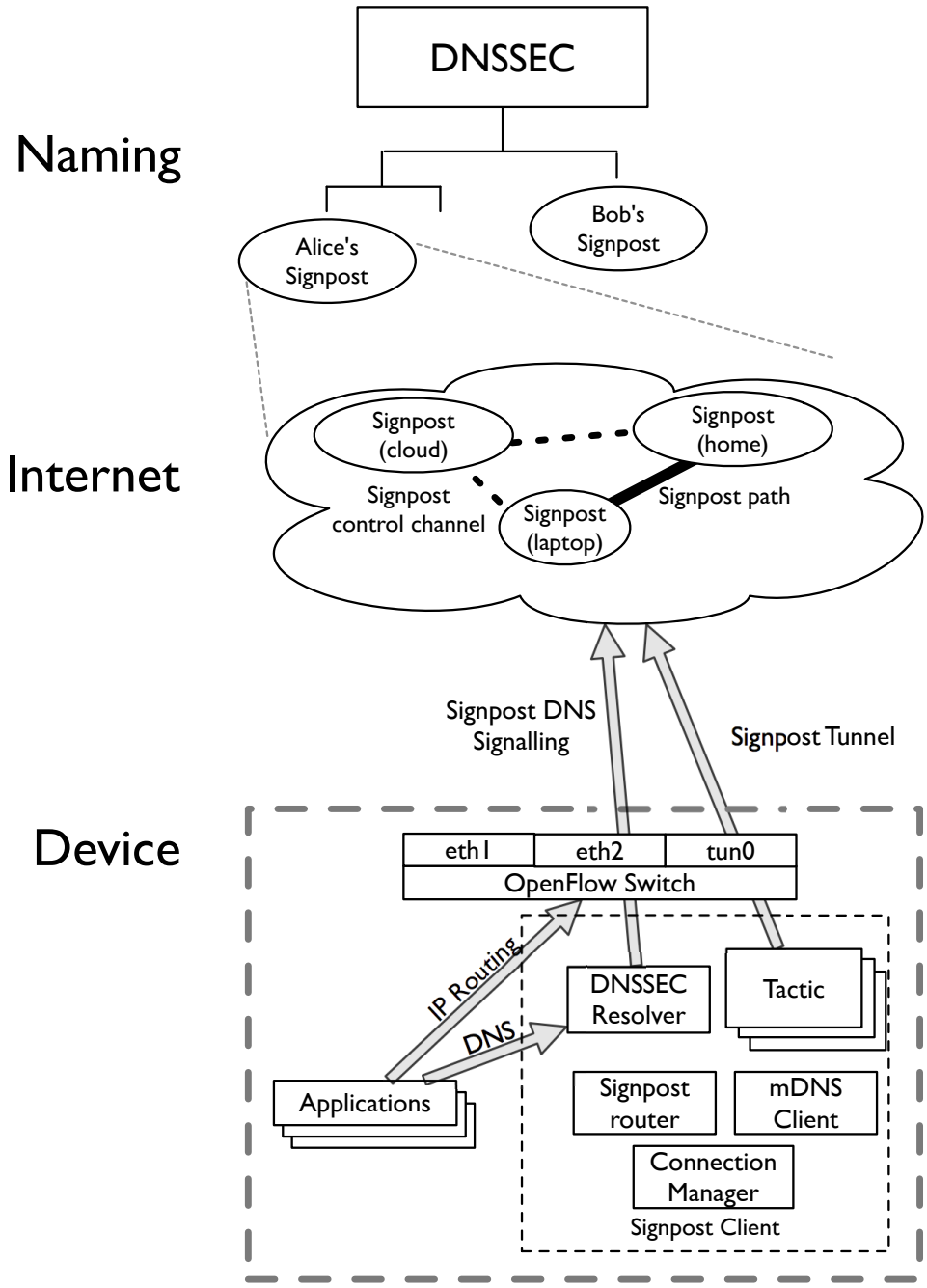


Figure 5.2: Signpost architecture presented from three different perspectives: The *user device*, the *Internet*, and the *naming hierarchy*.

Tactic name	Purpose	Protocol	Connectivity	Authenticate	Encrypt	Anonymize	Signpost
Avahi	Discover	UDP	-	No	No	No	Yes
Bonjour	Discover	UDP	-	No	No	No	Yes
UPnP	Discover	UDP	-	No	No	No	No
dns2tcp	Tunnel	DNS	TCP	No	No	No	No
DNScat	Tunnel	DNS	UDP	Yes	No	No	No
HTTP-Tunnel	Tunnel	HTTP	TCP	No	No	No	No
iodine	Tunnel	DNS	IP	Yes	No	No	Yes
NSTX	Tunnel	DNS	IP	Yes	No	No	No
Proxytunnel	Tunnel	HTTP(S)	TCP	Yes	Yes	No	No
ptunnel	Tunnel	ICMP	TCP	Yes	No	No	No
tuns	Tunnel	DNS	TCP	Yes	No	No	No
SSH	Tunnel	TCP	TCP/IP	Yes	Yes	No	Yes
IPSec	Tunnel	IP/UDP	IP	Yes	Yes	No	No
OpenVPN	Tunnel	TCP/UDP	IP	Yes	Yes	No	Yes
libjingle	Nat punch	TCP/UDP	UDP/TCP	Yes	Yes	No	No
privoxy	Anonymize	HTTP	HTTP/TCP	No	No	Yes	Yes
tor	Anonymize	TCP	TCP	No	Yes	Yes	Yes
stunnel	Encrypt	SSL	TCP	Yes	Yes	No	No
TCPCrypt	Encrypt	TCP	TCP	No	Yes	No	No

Table 5.1: List of available connection-establishing mechanisms. The table presents for each mechanism, its primary functionality, the protocol used to establish connectivity and the layer providing connectivity, and the support of the mechanism for Authentication, Encryption, Anonymization and Signpost integration.

5.3.1 Network Tactic

In order to provide end-to-end connectivity, Signpost uses available functionality from the wide range of free and open source connection-establishing mechanisms. Table 5.1 presents a survey of such mechanisms along with their network requirements and security properties, highlighting the significant diversity available between mechanisms. For example, some mechanisms provide connectivity, bypassing strict network policies, other mechanisms enhance security and privacy in end-to-end Internet paths, while a third class enables connection automation. Furthermore, available mechanisms vary significantly on the operating network layer and the exposed connection abstraction. Connection mechanisms expose connectivity over a specific transport layer port or through a network layer device. In terms of protocols, the majority use UDP or TCP sockets, but some mechanisms use other network protocols, like ICMP. Finally, authentication exhibits significant diversity, spanning from user-based authentication, using either passwords or certificates, to simple pre-shared passphrases, while some mechanisms lack support entirely.

In order to accommodate the variances between available connection-establishing mechanisms, Signpost defines a *Network Tactic* abstraction; a generic model encap-

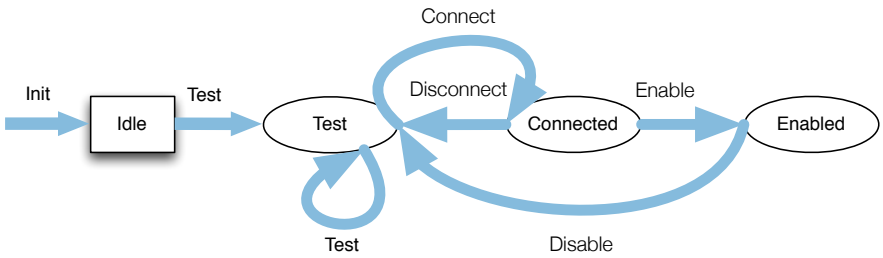


Figure 5.3: Signpost path lifecycle.

sulating testing and control requirements and automating the end-to-end path setup process for a connection mechanism. Each Network Tactic path is modelled as a four state automaton, presented in Figure 5.3. Path state is initialized in the *Idle* state. A test method invocation transfers the path state to the *Test* state and triggers Tactic-specific connectivity tests to discover the network policy and configuration requirements. If testing is successful, the path state can progress to the *Connected* state through an invocation of the connect method, responsible for configuring the underlying Tactic in order to establih the end-to-end path. Once the end-to-end path is established, the path state can progress to the *Enabled* state and forward traffic between the two devices over the path. Finally, the Tactic abstraction provides methods to backtrack from each state, tearing down any established path.

Furthermore, because Signpost Tactics require inter-device synchronisation and state distribution, their functionality is distributed between the Signpost controller and the Signpost devices, and split logically into two layers: the *Southbound layer* implements low-level Tactic operations and runs on Signpost clients; and the *Northbound layer* implements the state transitions of the path automaton, translating them into low-level operations, and runs on Signpost controllers.

The two layers of a Tactic employ the control channel between the devices and the Signpost controller to communicate using a JSON-RPC service [JSON-RPC Working Group et al., 2012]. The control channel is exposed as a TCP service on port 5353. Signpost uses a control channel with high availability to ensure Tactic parameter negotiation, even in heavily policed networks. If direct connectivity fails, the system uses as a fallback mechanism an Iodine [iodine, 2014] DNS tunnel. Iodine uses the DNS

protocol to transmit data between two hosts on UDP port 53, by encapsulating tunnel traffic in well-formed DNS packets which can travel through recursive DNS resolvers. Iodine can provide connectivity as long as the naming service, an important service for Internet connectivity, is not blocked by the network. The choice of Iodine provides sufficient capacity and latency to support the Signpost control channel, while better integration of control messages with the DNS packet format can improve the channel performance.

In order to present the separation between the two layers, we present the implementation details of the OpenVPN Tactic test method. For the specific functionality, the Southbound layer of the Tactic provides two operations: the *init* operation, initializing an OpenVPN server; and the *test* operation, testing client connectivity to an OpenVPN service. The Northbound test method uses Southbound operations to evaluate connectivity between two devices. During connection testing between two devices, the test method will execute in parallel on both devices: initially, the init operation and then the test operation towards the other device. The first device returning a successful test result becomes the client of the OpenVPN tunnel. If the test times-out or both devices return a negative result, then the controller will initialise a local OpenVPN server and execute the test operation on both devices towards the controller.

5.3.2 Network Routing

Signpost exposes connectivity on the network layer of the host, thus achieving backwards compatibility with existing decentralized applications. A Signpost cloud is abstracted as a local subnet and each device is assigned a persistent local IP. An agent-integrated ARP proxy replies with the MAC address `fe:ff:ff:ff:ff:ff` to all Signpost-local IP addresses, thus reducing broadcast traffic and focusing forwarding decisions on the network layer. Signpost relies on programmable network control mechanisms, like OpenFlow, to dynamically define forwarding policy and detect connection problems.

The majority of the Signpost control logic is defined by Network Tactics. Non-Signpost traffic is forwarded based on the default routing configuration of the host. For Signpost traffic, control is delegated to Tactics, which are responsible for installing appropriate forwarding rules. Tactics can inject and intercept traffic and exercise con-

trol pro-actively or reactively. Section 5.4.1 presents the integration details between Signpost and connection-establishing mechanisms, and elaborates on different network control approaches.

5.3.3 Connection Manager

Signpost Tactic management is implemented by the *Connection Manager* module. The module runs on the Signpost controller, controlling path establishment, selecting optimal paths between device pairs and ensuring connectivity. Path establishment uses the Tactic abstraction to isolate low-level connection details. In terms of path selection, Signpost can establish multiple end-to-end paths between two devices, but multipath connectivity is not supported by popular transport protocols, while newer multipath-enabled protocols, like SCTP [Ong and Yoakum, 2002] and multipath TCP [Ford et al., 2013], are not yet available in production systems. As a result, Signpost can use only a single end-to-end path between each pair of devices. Path search is initiated by the connect method of the Connection Manager module, using as parameters the device names and a list of security properties. Path selection takes under consideration two aspects: performance and security properties.

Path performance in Signpost reflects the resource availability in a path. The current implementation of Signpost uses a set of simple and static heuristics to predict path performance, based on the properties of the underlying Tactic. More specifically, Tactics using the Signpost controller as a path relay exhibit lower performance in comparison to direct paths, while Tactics using connectionless protocols, like UDP, are considered more efficient than Tactics using connection-oriented protocols, like TCP. These simple rules provide an approximation of the Tactic performance. Path performance characterisation can also incorporate active network measurements to improve flexibility and estimation accuracy.

In terms of security properties, Signpost considers three elementary functions for each Tactic: *encryption*, *authentication* and *anonymity*. An encrypted Tactic applies strong cryptography on both directions of the path, while an authenticated Tactic uses strong authentication during the establishment of the path. Finally, an anonymising Tactic provides built-in user identity obfuscation on the path. Each Tactic provides a subset of these functionalities, based on the capabilities of the underlying connection

mechanism (Table 5.1), and there is no single Tactic that supports all security properties, and ensures connectivity under any network environment. In order to address this limitation, Signpost defines a *Tactic synthesis* operation; a path that can be constructed by layering multiple connectivity mechanisms and effectively create a path providing the aggregate security properties. For example, an anonymised and authenticated path can be established using an SSH tunnel over a TOR circuit between the two devices. It is important to point out at this point that Signpost uses existing frameworks to enhance security for an end-to-end path and does not develop a security framework from scratch. The security properties of a path are a side effect of the used Tactics. End-users can use security properties to define security policies towards specific devices. The policy is expressed through a configuration file and users specify security requirements on a per domain basis.

Path selection is modelled as a dynamic optimization problem with security requirements modelled as constraints and path performance modelled as the objective function. Path performance is represented by a positive integer weight, with higher values reflecting lower performance of the path. The weight of a path is equal to the sum of the individual weights of the Tactics synthesizing the path. Path selection uses a breadth-first search mechanism over the complete space of all possible Tactic combinations. During a connection request the system spawns a thread for each Tactic, testing end-to-end connectivity. If the test is successful, the Tactic will try to connect the two hosts. If the connection is also successful, and there is either no other Tactic enabled or the enabled Tactic has a higher performance value, then the Manager will progress the state machine of the Tactic to the *Enabled* state, and disable any existing enabled Tactics. If the Tactic doesn't fulfil the security policy, the Manager will recursively try to layer more Tactics over the established path and find a Tactic synthesis with sufficient security properties. The Manager returns a successful result when the algorithm finds a path configuration fulfilling the security policy, but the search will continue to search for better paths. The search will terminate when all remaining Tactic combinations have a higher performance weight or they are synthesized by more than three Tactics. Effectively, the search algorithm rapidly establishes a first path between two devices, and progressively optimizes path performance while the devices remain interconnected.

Finally, the connection manager module is additionally responsible for monitoring

enabled paths and detecting disconnections. The module uses `flow_stats` OpenFlow messages to identify packet transmissions during a monitoring period, and thus infer path liveness. If the path is idle during that period the module uses a UDP heartbeat service to verify path connectivity. During a heartbeat timeout, the connection manager tears down the path and repeats the path search process. A Tactic can employ application-specific connectivity evaluation mechanisms, e.g. OpenVPN provides a keep-alive mechanism.

5.3.4 Effectful Naming

The majority of Internet-connected devices are essentially anonymous from a network perspective, being assigned transient names (e.g. via DHCP). The IPv6 protocol [Deering and Hinden, 1995] addresses a number of such limitations, but the protocol deployment on the edges of the Internet has been slow and limited. A fundamental goal for the Signpost architecture is to support stable and secure resolution of device names to concrete end-to-end paths. Signpost uses the DNS protocol [Mockapetris, 1987] to support its naming functionality for two reasons. Firstly, DNS is an effective solution to the naming problem. It is an Internet-wide service, supported by all network applications and never blocked in a network. The introduction of the DNSSEC protocol extensions provides additional bi-directional authentication. Secondly, the DNS service inherently supports delay-tolerant service indirection, used extensively across the Internet. Many services employ user-friendly domain names to represent virtual service end-points which are refined during name resolution, directing users to the best service mirror based on the end-host location and the service load [Barbir et al., 2003]. Effectively, a domain name resolution represents the interest of a user in connecting to a service, which the service provider can direct to a concrete end-point based on the network context.

The Internet naming service provides good scaling properties, using a hierarchical architecture. Every domain name consists of a sequence of name tokens organised in a hierarchical tree structure with a single root: the empty string. Every node on the tree can be coupled with a number of DNS Resource Records (RRs) and provides rich information on the domain name, like its network addresses and name aliases. In order to scale the performance of the naming service across the Internet, the design of the

system allows service delegation for a zone to other servers using the NS RR type. In addition, the DNS service defines a record caching mechanism, in order to improve lookup performance and service load.

A DNS packet (queries and responses) contains a standard header and four sections: the *question*, containing the name being looked up; the *answer*, containing RRs answering the question; the *authority*, pointing to an authoritative server for the question; and the *additional* records, containing any additional information pertaining to the question. Name resolution then follows one of two paths. A *recursive resolution* occurs when the server does not have the answer and so it acts as a resolver itself, pursuing the answer on behalf of the client. In contrast, an *iterative resolution* occurs when the server does not have the answer and responds by referring the client to a server "closer" to the answer.

Signpost reuses the DNS naming service abstraction but extends its functionality, introducing the idea of *effectful name resolution*. Effectful name resolutions have as a side-effect the creation of an end-to-end path towards the requested destination. Effectively, Signpost perceives name resolutions as an intention to connect to the requested device and translate the name resolution in an explicit connection request to the Connection Manager module. This extension to DNS functionality is similar to the indirection functionality of the protocol. In addition to pointing the requester to an appropriate network address, Signpost ensures a path towards that address.

Secure Names For All Internet Users The initial definition of the Internet naming service provided weak security guarantees and was vulnerable to man-in-the-middle and cache poisoning [Computer emergency response team, 2008] attacks. In order to enhance the security primitives, the IETF standardised a number of DNS protocol extensions, described as DNSSEC. DNSSEC defines four RR types [Arends et al., 2005], enabling response authentication as presented in Figure 5.4. In DNSSEC, each zone owns one or more cryptographic keys to sign authoritative DNS responses. The RRSIG RR type can carry RR signatures, while the NSEC RR type reflects the lack of specific RRs for a domain to a resolver. The DNSKEY RR type disseminates public keys of a domain to resolvers, which in turn can use them to authenticate response signatures. Finally, the DS RR type expresses the trust of a domain to a signing key of another domain. With respect to Figure 5.4, the domain `bob.signpost.io`

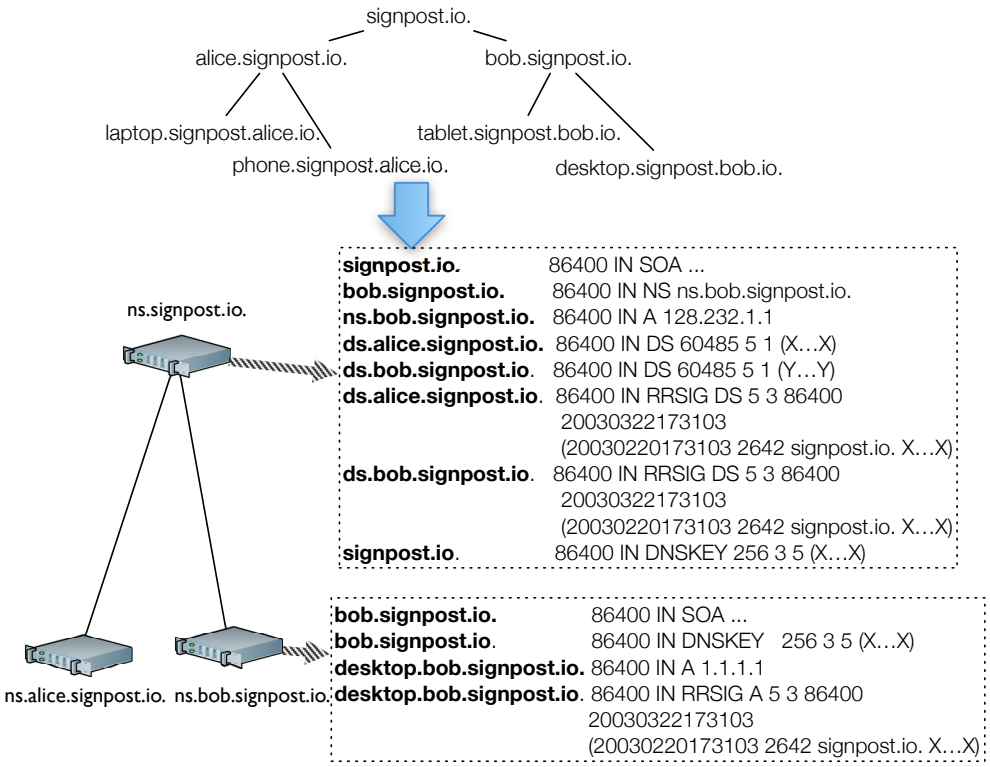


Figure 5.4: Example DNSSEC zone files for the `signpost.io` and `bob.signpost.io` domains. The `signpost.io` zone file contains a DS RR with the hash of the DNSKEY RR for `bob.signpost.io` and a RRSIG RR signature of the DS RR signed with the DNSKEY of the `signpost.io` domain. `bob.signpost.io` uses its DNSKEY to sign an A RR for `desktop.bob.signpost.io`.

signs an A record for the host desktop.bob.signpost.io through an RRSIG RR using Bob's DNSKEY. Bob's key is registered with the signpost.io domain through a DS RR, containing the hash of the key and signed using an signpost.io public key. Effectively, the DNSSEC extensions form a chain of trust as part of the naming service and ensure query response and key authenticity. The resolver requires, as in the X509 certificate architecture, only a list of authenticated anchors, configured out-of-band and injected into the authentication chain.

Signpost uses the DNSSEC RR types to establish an authenticated control channel between the devices of the user. Each Signpost user must have authoritative control of a domain zone, serviced by the Signpost controller, and register a signing key to the DNSSEC chain of trust. Each device has a domain name under the domain of the user. With respect to Figure 5.2, Bob is granted control of the zone bob.signpost.io where he registers his laptop under the domain name laptop.bob.signpost.io. As a result, each Signpost user has a globally accessible authenticated public identity for each device via a public-private key-pair. Using this key-pair, a user can sign and authenticate messages, bootstrap public key cryptography mechanisms and run key-exchange mechanisms, such as Diffie-Hellman-Merkle Rescorla [1999], and derive new shared private keys between any two devices over unsecure channels.

The base DNSSEC RR types provide a mechanism for authenticating responses from a nameserver to a DNS resolver. In terms of the Signpost architecture, we are additionally interested in authenticating DNS requests. Authenticated requests enable the server to present a different view over the resource mappings, depending on the querying entity[1]. Signpost uses the SIG(0) RR type, defined in [Eastlake 3rd, 2000], to sign DNS requests using the key of the device.

Signpost Integration With DNSSEC Signpost evolves the Signpost client and controller DNS functionality. Specifically, the controller runs a programmable DNS service, which is authoritative for the user domain. The server is authoritative for the user domain zone, signs responses on-the-fly, verifies SIG(0) signed queries, and translates them into Connection Manager requests. A request for an A RR for domain name laptop.alice.signpost.io, signed with a SIG(0) record with a key from host

[1] *"DNS servers can play games. As long as they appear to deliver a syntactically correct response to every query, they can fiddle the semantics."— RFC3234 [Carpenter and Brim, 2002]*

desktop.alice.signpost.io, will be translated into a connection request between Alice's laptop and desktop devices. In the client side of the Signpost architecture, we use a local Signpost-aware DNS resolver. DNS queries for non-Signpost hosts trigger a recursive search of the DNS name tree, in order to retrieve the requested RRs. For Signpost requests, the resolver signs the query with a SIG(0) record using the private key of the device.

DNS provides also service discovery functionality in a local network. Specifically, the DNS service discovery (DNS-SD) [Cheshire and Krochmal, 2013b] is an RR organisation specification which enables service advertisement and browsing in a local network. DNS-SD commonly uses the Multicast-DNS [Cheshire and Krochmal, 2013a] service and provides an efficient local service discovery mechanism, supported by most modern operating systems. Signpost advertises, using the DNS-SD mechanism, a Signpost service record with name _sp._tcp.local along with an RRSIG RR and the DNSKEY RR of the device, signed with the private key of the user domain. Another Signpost client can verify a Signpost service, using a locally cached copy of the Signpost controller public key, and establish a control channel. In offline mode, a query receiver functions as a Signpost controller, responsible for the server RR request for its domain name only.

5.3.5 Security and Key Management

Signpost extends the DNSSEC architecture to develop a public key distribution mechanism. Firstly, DNSSEC exhibits better trust chain integrity in comparison to SSL, which has a large set of root certificate providers, and supports incomplete trust chains via look aside validation [Weiler, 2007]. Secondly, a domain in DNSSEC has a single, well-defined owner, registered under a top-level domain, whereas URL-based identity schemes, like OpenID [Recordon and Reed, 2006], are subject to the trust to the domain owner.

Signpost key hierarchy extends the DNSSEC key hierarchy and introduces an additional layer of device keys in each user domain. DNSSEC deployment specifications dictate the use of at least two RR signing keys [Kolkman and Gieben, 2006]: A *Zone Signing Key (ZSK)* and a *Key Signing Key (KSK)*. ZSK signs the KSK, and its hash is stored as a DS RR in the top-level zone. KSK is used by the controller to sign

authoritative RRs. The use of two keys by the authentication mechanism provides a persistent anchor in the DNSSEC key hierarchy for each domain and enhances flexibility in key revocation; KSK rollover requires only a few record modification in the zone file. Signpost introduces an additional layer of *Device Signing Keys (DSK)*, which are signed using the KSK and used by the devices to bootstrap authentication. In order to join a Personal Cloud, a device must generate a DSK and add it in the zone file of the Signpost controller using a DNSKEY RR and a signed RRSIG RR. Any device with an anchor in the global DNSSEC key infrastructure can verify any Signpost signed request, by following the key chain in the DNSSEC tree.

5.4 Evaluation

In this section we evaluate a strawman implementation of Signpost and its compatibility with distributed Personal Cloud applications. Specifically: we present a strawman Signpost implementation and its integration with a set of Tactics (Section 5.4.1); we measure the performance of the available Signpost Tactics in a representative deployment scenario (Section 5.4.2); and we discuss the experience in the integration of Signpost with a set of popular applications (Section 5.4.3).

5.4.1 Signpost Implementation

Signpost is implemented predominantly in OCaml, a type-safe functional language providing enhanced security and fast prototyping. The implementation uses existing protocol libraries to integrate programmable protocol support for DNSSEC and OpenFlow. Our strawman implementation is fully functional under Linux and Android, using the Open vSwitch switch implementation, and MacOSX, using the ocaml-openflow library userspace switch implementation. The Signpost implementation supports the following connection-establishing mechanisms:

- *Direct*: The Direct Tactic evaluates the ability of two devices to connect directly without any tunnelling mechanism. The Tactic test method evaluates if direct bi-directional connectivity is possible between the two devices on a set of well-known service ports (e.g. TCP and UDP connectivity on ports 53, 80, 443 and

8080). If the tests are successful, the Tactic inserts OpenFlow rules on both devices to translate Signpost addresses into network addresses.

- *OpenVPN*: The OpenVPN Tactic integrates the respective tunnelling mechanism with Signpost. The Tactic test method evaluates inter-device and device-Signpost controller connectivity on UDP port 1194. The Tactic connect method uses the Signpost keys to bootstrap the OpenVPN authentication mechanism. Because of some limitations in OpenSSL certificate chain evaluation, for each path the Tactic generates transient private keys, signed by the user private key, and injects in the trusted certificate configuration of the OpenVPN software a certificate for the destination device key signed by its private key. The Tactic configures the OpenVPN software to expose connectivity through an Ethernet TAP interface [Krasnyansky et al., 2002], which is added to the local switch datapath. Each TAP device is assigned an IP in the 10.0.0.0/16 subnet and appropriate ARP announcement packets are broadcast, to bootstrap state in the internal OpenVPN ARP cache. Finally, The Tactic enable method inserts OpenFlow rules to forward packets over the OpenVPN tunnel and translate source and destination addresses, similarly to the direct Tactic.

- *SSH*: The SSH Tactic provides inter-device connectivity using the SSH protocol. The Tactic configures and runs an SSH server on every Signpost device on port 10000, configured exclusively to provide tunnelling functionality using key-based authentication. The Tactic spawns a separate SSH daemon on each device, to enable tighter service configuration. Tactic testing evaluates direct TCP connectivity between the devices or through the Signpost Controller. The connect method of the SSH Tactic appends the public key of the destination device in the `authorized_key` and the `known_host` SSH configuration files and uses the device private key to authenticate the client and server during connection. The SSH Tactic associates the SSH tunnel with a local TAP Ethernet interface on each device and the Tactic inserts appropriate OpenFlow rules to forward traffic to the TAP interface.

- *Privoxy*: The Privoxy Tactic enables HTTP request anonymisation using the Privoxy [2013] HTTP proxy. The Tactic provides a mechanism for purging

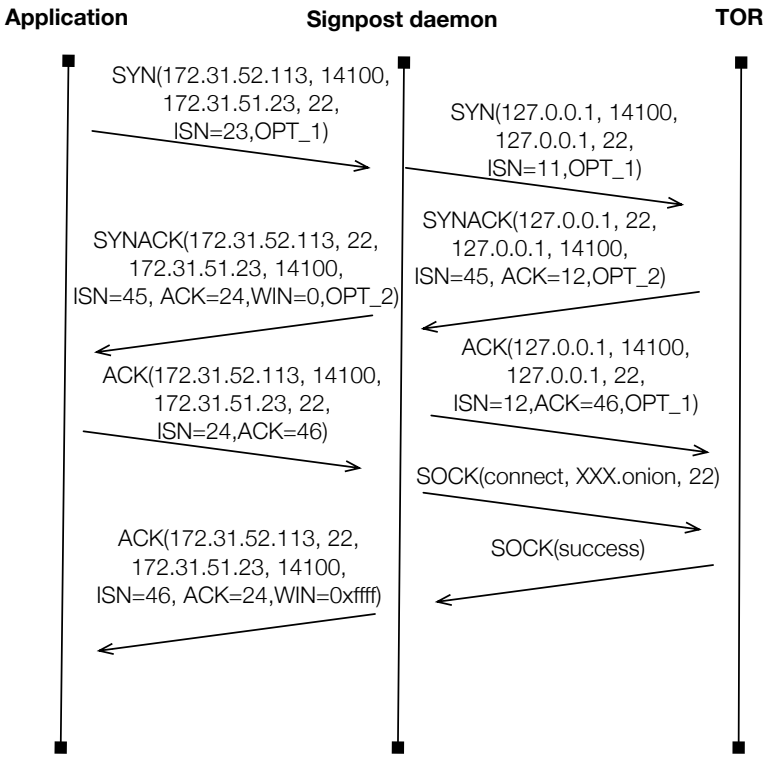


Figure 5.5: TOR Tactic connection establishment sequence diagram. The Tactic translates each TCP connection request to a Signpost device into a SOCKS request to the TOR proxy.

HTTP requests and responses from user identification elements, but does not ensures connectivity. The Tactic test method configures a Privoxy HTTP proxy server on each device, configured with a strict anonymisation policy, and succeeds only when the Tactic is used in a synthesised path and the Tactic is tested over an existing connection-establishing Tactic (e.g. direct, OpenVPN, etc.). The Privoxy connect method handles TCP flows in a proactive manner. For each SYN packet, the Tactic installs bi-directional flows, forwarding data from the application to the local listening port of the Privoxy daemon.

- *TOR*: The TOR Tactic enables connectivity over the TOR anonymised network [Dingledine and Mathewson, 2006]. The Tactic configures a TOR client on each device, exposing a SOCKS proxy and uses the TOR hidden service functionality to support inter-device connectivity. The Tactic generates for each host an

anonymised domain name under the `.onion` domain and employs the embedded name lookup capability of the SOCKS protocol to provide host discoverability over end-to-end TOR circuits. The TOR test method initializes the TOR client, propagates the domain name of the device to the Signpost controller and tests the effectiveness of the Tactic in the network environment.

The TOR Tactic uses a reactive control scheme. For each SYN packet to a Signpost host, the Tactic injects packets to initiate a TCP connection with the local SOCKS proxy and sends a SOCKS request to establish a TOR circuit to the destination device. On TCP connection establishment with the local SOCKS proxy, the Tactic responds to the initial SYN request with a SYNACK packet with sequence and ACK numbers which accommodate the additional bytes of the SOCKS request, and a zero receive window, to suppress any further data transmissions from the application. Once a positive SOCKS response is received, the Tactic sends to the initiating TCP flow an ACK packet with a non-zero window and insert appropriate OpenFlow rules for in-kernel IP address and port number translation. On negative SOCKS response, the Tactic injects a TCP RST packet and tears down the flow. Figure 5.5 presents a sequence diagram of the Tactic interactions. In order to replicate a similar abstraction for UDP and ICMP traffic, we setup a VTun [2012] tunnel over TOR, thus enabling IP-level connectivity. We avoided using the tunnel for TCP connections in order to reduce header capacity losses of the tunnelling mechanism.

- *DNS-SD*: The DNS-SD Tactic enables service advertisement between Signpost devices. DNS-SD [Cheshire and Krochmal, 2013b] is a common OS service enabling hosts to advertise services in the local network. The Tactic aids existing decentralised Personal Cloud applications to discover available services running on other Signpost devices. Similarly to Privoxy, the Tactic does not provide connectivity and thus can only be used in synthesized paths. The Tactic connect method intercepts DNS-SD packets from the network stack of the device and propagates them over the control channel to the other devices of the user. Signpost daemons inject equivalent DNS-SD multicast packets through the OpenFlow protocol to the network loopback device and thus provide service discovery.

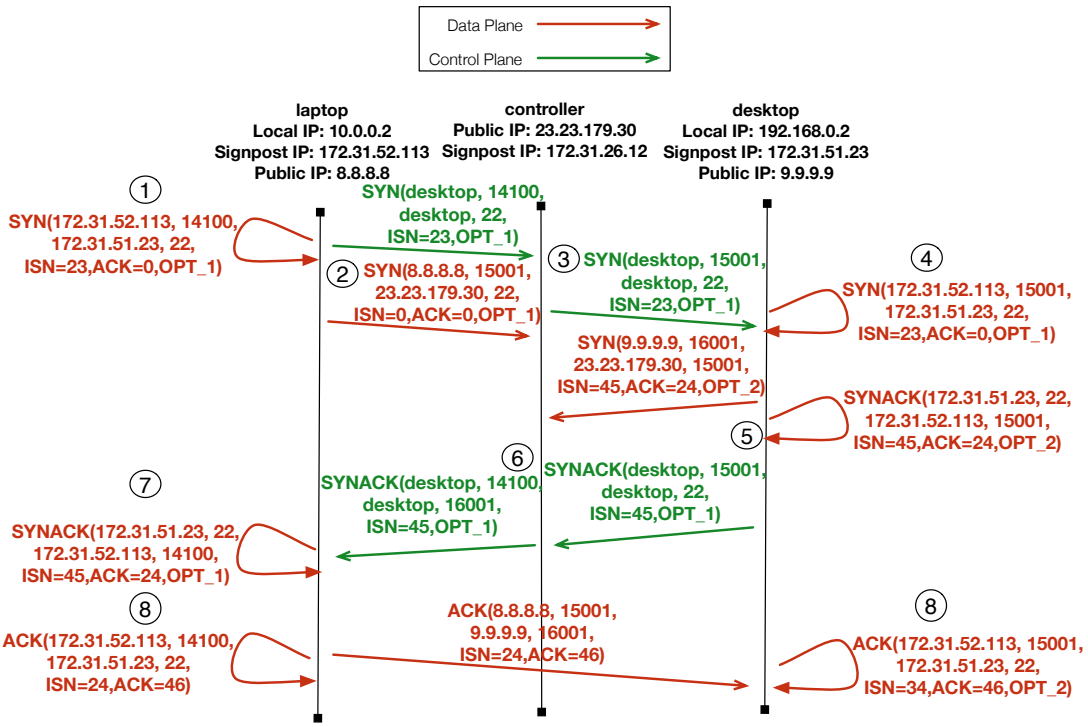


Figure 5.6: NAT-punch Tactic connection establishment sequence diagram. The Tactic injects crafted packets to create appropriate state in the local NAT, while the controller detects the applied port mapping for both hosts of the Signpost path.

- *NAT-punch*: The Tactic implements a packet injection technique which allows TCP and UDP flows to bypass NAT boxes. The Tactic supports only Full-cone NAT, Address-restricted NAT and Port-restricted NAT [Rosenberg et al., 2003]. The NAT-punching Tactic uses the Signpost controller to infer the port mappings configuration of the local NAT. We present a sequence diagram between two devices and the Signpost controller in order to establish a TCP connection using the NAT-punch Tactic 5.6. Specifically, for a new TCP connection, the daemon intercepts the SYN packet (Step ①), propagates important header fields (Port number, initial sequence number and TCP options) over the control channel to the remote device and sends a SYN packet from the device to the Signpost controller, in order to infer the port mapping policy and create the appropriate state in the NAT (Step ②). When the Signpost controller receives the SYN, it extracts the source port mapping applied by the NAT and forwards it, along with the TCP initial sequence number, to the destination device (Step ③). Once the destination device receives the control message from the Signpost controller, it will generate a SYN packet, using the header fields of the initial SYN packet, and send both to the listening service and the Signpost Controller (Step ④). In the destination device, the Signpost daemon intercepts the SYNACK response of the listening service and propagates the TCP header fields over the control channel to the connection initiating device (Step ⑤). Once the exact port-mapping is inferred by the Controller, the information is propagated to the initiating device (Step ⑥), which will inject a SYNACK packet to the local stack to progress the TCP-handshake (Step ⑦). Finally, once the two devices know the NAT port-mapping and have created the appropriate state in the NAT flow table, the Tactic inserts OpenFlow rules to translate appropriately incoming and outgoing traffic of the flow (Step ⑧).

  For UDP traffic, the Tactic controller will intercept the initial UDP packet of the flow and propagate it over the control channel to the remote device, as well as transmit a similar packet to the Signpost controller in order to generate the appropriate state in the intermediate NAT services. The remote device sends a similar UDP packet to the Signpost controller. Once the controller has received both UDP packets, it will notify both devices to appropriately transform flow IP address and port numbers, using OpenFlow flows.

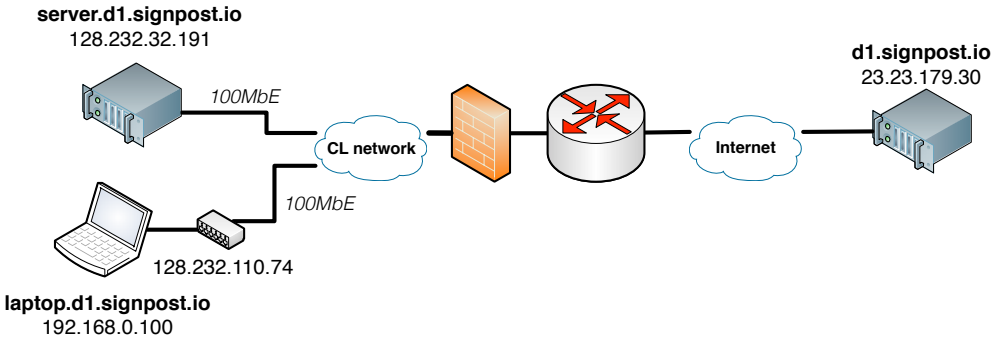


Figure 5.7: Signpost Tactic evaluation topology. *server* and *laptop* hosts are connected on different subnets of the CL network. The laptop is connected to the network through a NATed home router and a stateful firewall enforces network policy between the two subnets. The Signpost *controller* is hosted on an Amazon EC2 micro instance.

Tactic name	Direct Connectivity			Cloud-assisted Connectivity		
	setup (sec)	throughput (Mbps)	RTT (msec)	setup (sec)	throughput (Mbps)	RTT (msec)
Direct	1.0	94.1	1.5	-	-	-
OpenVPN	13.3	66.0	1.23	20.0	1.9	172.0
SSH	6.8	86.5	2.73	8.8	1.7	829.0
TOR	60.2	2.4	912.0	-	-	-
NAT-punch	1.2	94.1	1.5	-	-	-

Table 5.2: Signpost Tactic performance evaluation in terms of latency to setup a path, throughput and RTT delay. Direct cloud-assisted paths exhibit two orders of magnitude higher performance and lower throughput.

5.4.2 Tunnel Evaluation

In order to evaluate the performance of our strawman implementation, we conducted a series of measurements for each Tactic. Our testbed consists of three hosts, two Signpost devices and a Signpost controller, presented in Figure 5.7. The Signpost clients (`laptop.d1.signpost.io`, `server.d1.signpost.io`) are connected to the production network of the Computer Laboratory on different subnets through 100 MbE links. The network enforces a security policy between both devices and towards the Internet using a stateful firewall. Additionally, the host laptop is connected to the network through a home router with NAT functionality. The Signpost controller of

our experiment runs on an Amazon EC2 micro instance [Amazon, 2014]. Using Signpost, we establish all possible end-to-end Tactics[1] and measure the path setup delay, the achievable throughput, and the round trip delay for each Tactic. More specifically, in each experiment we initialize the Signpost daemon on each device, and establish connectivity using a specific Tactic. For each established path we measure the delay in completing a Signpost name resolution and initialize a Signpost path, the path throughput, using a steady-state TCP measurement probe, and the average path RTT, using a 1 Mbps UDP measure probe. We conducted our measurements ten times for each possible configuration and present the median values in Table 5.2.

From the results of our experiment, we note the difference in capacity and latency when a path uses the Signpost controller. Edge network connectivity provides two to three orders of magnitude higher performance in comparison to cloud-assisted paths. In addition, we highlight the trade-off between security and performance between Tactics. For example, the TOR anonymisation functionality significantly degrades performance and latency, deeming it inadequate for responsive applications. The performance degradation is primarily due to the onion routing scheme used to obfuscate the source and destination addresses of the flow, and the sharing nature of the system. Signpost exposes a clear control abstraction of the trade-off between performance and security.

An important aspect of Tactic performance is the path setup latency. This delay is important because the DNS protocol employs time-outs in name resolutions to detect unresolvable domain names. Linux and OSX have a maximum delay of 15 seconds[2] (3 retries with 5 seconds time-out per request), while for Windows the delay varies between 12 to 15 seconds depending on the OS version[3]. Bootstrapping a Signpost path incurs noteworthy setup latencies, but the DNS naming service is designed to handle such latencies. Based on Table 5.2 results, the majority of the Tactics exhibits setup latencies within the resolver time-out limit, except the TOR Tactic. TOR setup latency is dominated by the circuit establishment latency of the onion overlay. The current implementation exposes the setup latency to applications and thus may experience DNS

[1] We exclude the privoxy and DNS-SD Tactics because they improve functionality on established paths only.

[2] http://linux.die.net/include/resolv.h

[3] http://blogs.technet.com/b/stdqry/archive/2011/12/15/dns-clients-and-timeouts-part-2.aspx

name lookup time-outs for the first flow of a path. An optimistic approach can mitigate potential resolver time-outs (e.g. use of DNS aliases to delay the response). It is important at this point to clarify that the measured setup latencies occur only for the first flow of a path, and subsequent flows will experience name resolution delays equal to the round-trip time to the Signpost controller.

Finally, using the aforementioned topology we also measured the performance of Tactics synthesis. The results of the experiments report that the throughput and RTT of such paths is equal to the minimum value between the Tactics forming the network path, while the setup latency is equal to the sum of the individual Tactics.

5.4.3 Application Compatibility

In this section we evaluate the compatibility of Signpost with applications providing resource and information sharing. Our evaluation focuses on the effectiveness of Signpost integration with such applications. We used the topology in Figure 5.7 and considered two primary classes of applications: *File Sharing* and *Resource sharing*. In terms of file sharing, we tested three popular applications, namely the Linux NFS implementation [Shepler et al., 2010], git-annex [git annex, 2013] over rsync and SSH Secure Copy. In terms of resource sharing we tested the functionality of CUPS remote printing, DAAP media sharing, VNC remote desktop and SSH remote access.

We report that all applications functioned correctly from a network perspective over all Tactic configurations. Nonetheless, user-perceived performance is proportional to the underlying Tactic performance (Table 5.2). Some Tactics incur significant RTT delays and degrade user experience in reactive applications. For example, the SSH software exhibited significant responsiveness problem when routed over TOR. In addition, through the integration testing we identified limitations in the openness of our architecture and modified accordingly our implementation. For example, we observed that the Linux SSH client and server use different values for the ToS bits, depending on the payload of the packet, and require the respective OpenFlow rule to wildcard the ToS field value in the NAT-punch and TOR Tactics. Our design choice to integrate Signpost with existing network applications in the network layer of the OS is effective and enables seamless integration with existing network applications.

5.5 Summary

This chapter focused on the problem of Internet-scale inter-device connectivity. Our work is motivated by the end-user requirement for resource and information sharing services using network device federation between its personal devices; a generic abstraction which we termed Personal Cloud. We elaborated on the available mechanisms and highlighted the trade-offs among them, identifying a significant connectivity and naming problem in the current Internet architecture. We argued that the abundance of connection-establishing mechanisms provides a sufficient toolset to connect devices across the Internet, but requires a distributed control framework to orchestrate the automatic establishment of end-to-end paths using these mechanisms. We presented Signpost, a distributed naming and connectivity framework which enables global device naming, automates the testing and configuration of end-to-end paths and provides secure and backwards-compatible connectivity to existing network applications.

Signpost evolves two aspects of user device control plane. Firstly, the control plane is extended to accommodate *Network Tactic* functionality; mechanisms enabling ad-hoc end-to-end connectivity, like tunnelling software and NAT punching techniques. The architecture develops a generic model coupled with a secure and high-availability Internet-wide control channel, enabling inter-device Tactic testing and configuration orchestration. In addition, the rich security properties provided by existing network Tactics, allow the system to expose a security control abstraction, providing trade-off control between performance and security. Secondly, the architecture integrates the control plane logic with the naming service. Signpost provides Internet-wide persistent device names and a global and secure key distribution mechanism, building on top of the DNSSEC extensions.

We presented our strawman Signpost implementation and its integration with network connectivity applications, and evaluated the performance of Signpost paths and its compatibility with existing network applications. We elaborated on the integration details of Signpost with direct, OpenVPN, TOR, SSH, Privoxy, DNS-SD and NAT-punch software and connection techniques, supporting the generality of the Signpost Tactic model. Additionally, we measured the performance of integrated Tactics in a typical deployment scenario and verified the backwards-compatibility of the system with a number of popular decentralised network applications. In the next chapter, we

conclude the results of our research and discuss future work directions.

Chapter 6

Conclusions and Future Work

This chapter concludes the dissertation by summarising the work it described, and noting areas in which further work is required.

6.1 Summary

This dissertation has addressed issues of network performance scalability using the SDN paradigm. Chapter 1 began by motivating the requirement to scale the functionality of existing network protocols and technologies in order to support the design limitation which are highlighted by the increase in network technology adoption. We argued that network performance is multi-dimensional (e.g. resource control, management, connectivity), and depends on the applications and the deployment environment. We claimed that specialized control plane architectures can mitigate network bottlenecks introduced by protocol design, while ensuring backwards compatibility and high performance. Effective control plane evolution must address the requirements and exploit the inherent opportunities of the deployed environment.

Chapter 2 then considered background and related work to the problem of network control. We provided a bottom-up design discussion on available network control mechanisms. We elaborated on the architecture of current network devices and presented a generic architecture model of the integration between the control and data plane in a single device in order to highlight the physical limitations in network performance. Furthermore, we reviewed the current production-level network control protocols and mechanisms for the data link and network layers and it was argued that their

ability for responsive, flexible, and user friendly control is limited. Furthermore, we surveyed a series of experimental approaches which address network control limitations and provide flexible, distributed and evolvable control. Namely we presented Active Network, Devolved Control of ATM Networks and Software Defined Networking. In order to highlight the opportunities provided by these mechanisms, we concluded this chapter with an extensive presentation of novel control applications build on top of the SDN paradigm.

The bulk of the experimental contribution of this dissertation was reported in the following three chapters. Chapter 3 analysed the elementary scalability of the SDN paradigm. In this chapter, we presented two measurement platforms, enabling network experimenters to evaluate the performance of SDN architectures. OFLOPS is a high-precision hardware-accelerated OpenFlow switch measurement platform which enables experimenters to understand the performance behaviour of OpenFlow-enabled devices, and SDNSIM, a lightweight high-precision network simulation and emulation framework. Using OFLOPS, we developed and ran a series of tests characterising the elementary functionalities of the OpenFlow protocol and detected significant variance between switch protocol implementations. SDNSIM is a high precision experimentation environment, which allows users to implement their control logic and traffic models and evaluate the performance of an SDN architecture. The platform provides the ability to simulate, using the ns-3 platform, or emulate, using the Xen virtualisation platform, a experimental definition. The platform provides enhanced realism on the performance characteristics of the network control plane, while through our evaluation we highlighted the achieved experimental scalability. Using SDNSIM, we replicated the functionality of a small-scale datacenter network and evaluated the effectiveness of control centralisation.

Chapter 4 elaborated on the problem of management scalability in modern home networks. Using ethnographic and measurement studies of the home setting, we identified significant mismatches between the user-requirement and user-understanding of network functionality, and the existing technologies. Motivated by this observation, we redesigned the home router, implementing a series of control modifications which enhanced user control and bridged the user perception with the underlying network functionality. The proposed architecture was extended with a collaborative resource control mechanism which integrated user application-level requirements with the ISP

policy, in an effort to develop a user-friendly control scheme for the last-mile bottleneck in residential broadband networks. We presented the development of a strawman implementation of our system and verified that: the architectures incurs minimal impact on network functionality; the protocol modifications remain backwards compatible with a number of popular devices, OSes and applications; and the resource control mechanism improves the support for latency and bandwidth sensitive applications in congested residential networks.

Chapter 5 discussed Internet-scale naming and connectivity scalability. Through the work of this chapter, we aimed to develop a decentralised and Internet-wide federated network between the devices of a user. We motivated our effort by presenting the limitations of existing approaches in terms of usability and privacy, and argued that an evolved control plane for end-hosts can address such limitations and support all required functionality. We proposed the Signpost architecture providing secure, continuous and decentralised inter-device connectivity. Signpost reuses existing connectivity mechanisms to provide ad hoc end-to-end paths between devices. The system provides a global naming structure and uses the DNS protocol to establish a global control channel through which Signpost automates distributed evaluation and configuration of connection-establishing mechanisms. We presented a strawman implementation of Signpost and its integration with OpenVPN, TOR, SSH, Privoxy, NAT punching and DNS-SD mechanisms. Using the Signpost implementation, we characterised the low impact of the architecture on network functionality and its backwards compatibility with existing applications.

6.2 Contributions

The dissertation makes the following three contributions:

Control plane scalability This thesis presented the first scalability characterisation of OpenFlow implementations. We highlighted the significant performance diversity between implementations which can affect the performance and the correctness of control architectures. Furthermore, we developed SDNSIM, an experimentation platform for SDN architectures, providing the ability to emulate and simulate network experi-

ments. SDNSIM provides high fidelity and scalability on replicating complex network control architecture and provides intuitive control between fidelity and scalability. Using SDNSIM we evaluate the performance of a hierarchical control architecture in a small-scale datacenter.

Management scalability This thesis presented a novel control architecture establishing scalable management for home network. We presented a flow-based controller, which exploited the social conventions in the home to manage introduction of devices to the network, and their subsequent access to each other and Internet hosted services. Additionally, we proposed a modification in the control architecture of the ISP network, which enables users to express and enforce their resource requirements in a user-friendly manner. We provided strong evidence on the scalability, backwards compatibility and effectiveness of our solution.

Connectivity and naming scalability This thesis analysed the significant limitation introduced by the current Internet architecture on the connectivity ability between the devices of users. In order to mitigate these limitations we presented Signpost, a decentralised control architecture providing global names for user devices and continuous connectivity between them. We presented the flexibility of Signpost in encapsulating a wide range of connection-establishing mechanisms and provided strong evidence on its backwards compatibility with existing applications, and its performance scalability.

6.3 Future Work

The experimental results and the practical solutions presented in the thesis provide fruitful seeds to cultivate a wider research agenda on control plane evolvability.

6.3.1 Distributed Network Control

One of the first use cases of the SDN paradigm was the centralisation of control in order to improve policy effectiveness and ease network management. Nonetheless this

vision has evolved and refocused on the development of distributed control architectures. Control distribution is motivated by two observations. Firstly, load and latency of the control channel increases proportionally to the size of the network, while reliability guarantees relax. Secondly, defining one global control policy which is able to encapsulate multiple policy aspects (e.g. security, performance, access control) exhibits significant complexity. Applications have shifted to a multi-controller paradigm using either centralised proxies Sherwood et al. [2010b] or separating the network in domains and using distribute algorithms to synchronise state between controllers Koponen et al. [2010].

The work presented in Chapter 3 provides a scalable control experimentation platform, a powerful tool to understand further the impact of distributed design patterns on the performance of a network. Hierarchical control, as presented through the evaluation experiment using the SDNSIM, has low impact on network performance, but a complete architectural design requires further evaluation of mechanisms with strong control plane responsiveness and reliability guarantees.

6.3.2 User-centric Networking

As we have discussed in Chapter 4, current network technologies exhibit a significant mismatch with the user requirements. The outcomes from the previous two chapters have provided strong evidence on the ability of networks to reconsider control and augment it with meaningful user input. Rather than relegating users to an artefact of the application layer, accommodating users and their relationships at all layers of the system can improve user satisfaction and network functionality. We consider two main extensions on the work of the thesis towards this goal.

Firstly, a number of areas arise from the work in Chapter 4. The presented control architecture provides novel opportunities to augment home network functionality and exploit the wider social context of the home setting. In the local network, home guests can inject their configuration into the local network policy and improve their experience. This approach is not limited in the local network and can expand to wider contexts. For example, neighbours can negotiate resource control by taking advantage of the social context in a neighbourhood. Users can coordinate socially to share un-utilized resources on the last mile of the residential connection, e.g. exchange high

priority traffic for specific timeslots with neighbours.

Secondly, a number of pieces of work arise from Chapter 5. The Signpost design at the moment is limited in its policy expressiveness, but is sufficient to address our motivations. Nonetheless, the authentication primitives can provide a scalable and global mechanism to develop novel network policy frameworks and address a series of problems stemming from the inability of the network layer to authenticate network end-points. Furthermore, the hierarchical structure of Signpost can be used to reflect higher level social relationship and increase the in-network flexibility. Such a control architecture can take advantage of recent theory frameworks, like Bigraphs, to scale the respective complexity.

References

Luigi A., Antonio I., and Giacomo M. The Internet of Things: A survey. *Computer Networks*, 54(15):2787 – 2805, 2010. 103

B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Ed. Levkowetz. Extensible Authentication Protocol (EAP). RFC 3748, IETF, June 2004. 86

A. Akella, S. Seshan, and A. Shaikh. An empirical evaluation of wide-area internet bottlenecks. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, IMC '03, pages 101–114, Miami Beach, FL, USA, 2003. ACM. 93

M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, SIGCOMM '08, pages 63–74, Seattle, WA, USA, 2008. ACM. 69

M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: dynamic flow scheduling for data center networks. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, NSDI'10, pages 19–19, San Jose, California, 2010. USENIX Association. 35

D S. Alexander, B. Braden, Carl A Gunter, Alden W Jackson, Angelos D Keromytis, Gary J Minden, and D. Wetherall. Active Network Encapsulation Protocol (ANEP). Experimental, 1997a. 21

D. S. Alexander, M. Shaw, S. M. Nettles, and J. M. Smith. Active bridging. In *Proceedings of the ACM SIGCOMM '97 conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '97, pages 101–111, Cannes, France, 1997b. ACM. 21

D. S. Alexander, W. A. Arbaugh, M. W. Hicks, P. Kakkar, A. D. Keromytis, J. T. Moore, C. A. Gunter, S. M. Nettles, and J. M. Smith. The SwitchWare active network architecture. *Netwrk. Mag. of Global Internetwkg.*, 12(3):29–36, May 1998a. 21

D.S. Alexander, W.A. Arbaugh, A.D. Keromytis, and J.M. Smith. Safety and security of programmable network infrastructures. *Communications Magazine, IEEE*, 36 (10):84–92, 1998b. 21

Amazon. EC2 Instances. http://aws.amazon.com/ec2/instance-types/, 2014, Online; accessed February 2014. 129

L. Andersson, I. Minei, and B. Thomas. LDP Specification. RFC 5036 (Draft Standard), October 2007. URL http://www.ietf.org/rfc/rfc5036.txt. 18

A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15(2):121–142, June 2006. 79

R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Resource Records for the DNS Security Extensions. RFC 4034 (Proposed Standard), March 2005. URL http://www.ietf.org/rfc/rfc4034.txt. 118

P. Arlos and M. Fiedler. A method to estimate the timestamp accuracy of measurement hardware and software tools. In *Proceedings of the 8th International Conference on Passive and Active Network Measurement*, PAM'07, pages 197–206, Louvain-la-Neuve, Belgium, 2007. Springer-Verlag. 40

G. Armitage, M. Claypool, and P. Branch. *Networking and online games: understanding and engineering multiplayer Internet games*. Wiley. com, 2006. 5

D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow. RSVP-TE: Extensions to RSVP for LSP Tunnels. RFC 3209 (Proposed Standard), December 2001. URL http://www.ietf.org/rfc/rfc3209.txt. 9, 18

G. Balestra, S. Luciano, M. Pizzonia, and S. Vissicchio. Leveraging Router Programmability for Traffic Matrix Computation. In *Proceedings of the Workshop on*

Programmable Routers for Extensible Services of Tomorrow, PRESTO '10, pages 11:1–11:6, Philadelphia, Pennsylvania, 2010. ACM. 52

J. R. Ballard, I. Rae, and A. Akella. Extensible and scalable network monitoring using OpenSAFE. In *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, INM/WREN'10, pages 8–8, San Jose, CA, 2010. USENIX Association. 32

A. Barbir, B. Cain, R. Nair, and O. Spatscheck. Known Content Network (CN) Request-Routing Mechanisms. RFC 3568 (Informational), July 2003. URL http://www.ietf.org/rfc/rfc3568.txt. 117

P. Barford and M. Crovella. Generating Representative Web Workloads for Network and Server Performance Evaluation. In *Proceedings of the 1998 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '98/PERFORMANCE '98, pages 151–160, Madison, Wisconsin, USA, 1998. ACM. xi, 99

P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. M. Pratt, and A. Warfield. Xen and the Art of Virtualization. *SIGOPS Oper. Syst. Rev.*, 37(5):164–177, October 2003. 57

T. Bates, E. Chen, and R. Chandra. BGP Route Reflection: An Alternative to Full Mesh Internal BGP (IBGP). RFC 4456 (Draft Standard), April 2006. URL http://www.ietf.org/rfc/rfc4456.txt. 19, 33

R. Bellman. On a routing problem. Technical report, DTIC Document, 1956. 19

T. Benson, A. Anand, A. Akella, and M. Zhang. MicroTE: fine grained traffic engineering for data centers. In *Proceedings of the 7th COnference on emerging Networking EXperiments and Technologies*, CoNEXT '11, pages 8:1–8:12, Tokyo, Japan, 2011. ACM. 35

B. Beranek. A History of the ARPANET: The First Decade. http://www.darpa.mil/WorkArea/DownloadAsset.aspx?id=2677, 1981. 4

G. Berger. NodeFlow: An OpenFlow Controller Node Style. http://garyberger.net/?p=537, 2012, Online; accessed February 2014. 31

A.K. Bhushan. File Transfer Protocol. RFC 354, July 1972. URL http://www.ietf.org/rfc/rfc354.txt. 4

A.K. Bhushan, K.T. Pogran, R.S. Tomlinson, and J.E. White. Standardizing Network Mail Headers. RFC 561, September 1973. URL http://www.ietf.org/rfc/rfc561.txt. 4

A. Bianco, R. Birke, L. Giraudo, and M. Palacin. OpenFlow Switching: Data Plane Performance. In *Proceedings of the 2010 IEEE International Conference on Communications (ICC)*, pages 1–5, May 2010. 43

S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. RFC 2475 (Informational), December 1998. URL http://www.ietf.org/rfc/rfc2475.txt. 9

R. Braden. Requirements for Internet Hosts - Communication Layers. RFC 1122 (INTERNET STANDARD), October 1989. URL http://www.ietf.org/rfc/rfc1122.txt. 4

British Telecoms. Broadband usage policy. http://bt.custhelp.com/app/answers/detail/a_id/10495/~/broadband-usage-policy, 2013. 94

Broadband Canada Program. Broadband Canada: Connecting Rural Canadians - Frequently Asked Questions. http://www.ic.gc.ca/eic/site/719.nsf/eng/h_00004.html, 2013. 94

Broadcom. Switching silicon chips. http://www.broadcom.com/products/Switching/Data-Center, 2013, Online; accessed February 2014. 14

M. A. Brown. Pakistan hijacks YouTube. http://www.renesys.com/blog/2008/02/pakistan_hijacks_youtube_1.shtml, 2008, Online; accessed February 2014. 20

P. Brundell, A. Crabtree, R. Mortier, T. Rodden, P. Tennent, and P. Tolmie. The Network from Above and Below. In *Proceedings of the 1st ACM SIGCOMM Workshop on Measurements Up the Stack*, W-MUST '11, pages 1–6, Toronto, Ontario, Canada, 2011. ACM. 75

M. Butkiewicz, H. V. Madhyastha, and V. Sekar. Understanding website complexity: measurements, metrics, and implications. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, IMC '11, pages 313–328, Berlin, Germany, 2011. ACM. 5

M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe. Design and Implementation of a Routing Control Platform. In *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, pages 15–28. USENIX Association, 2005. 19

Z. Cai, A. L. Cox, and T. S. E. Ng. Maestro: balancing fairness, latency and throughput in the OpenFlow control plane. Technical Report TR11-07, Rice University, 2011. URL http://www.cs.rice.edu/~eugeneng/papers/Maestro-TR11.pdf. 31, 63

K. L. Calvert, W. K. Edwards, N. Feamster, R. E. Grinter, Y. Deng, and X. Zhou. Instrumenting home networks. In *Proceedings of the 2010 ACM SIGCOMM workshop on Home networks*, HomeNets '10, pages 55–60, New Delhi, India, 2010. ACM. 35

M. Canini, D. Venzano, P. Perešíni, D. Kostić, and J. Rexford. A NICE way to test openflow applications. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, NSDI'12, pages 10–10, San Jose, CA, 2012. USENIX Association. 35

B. Carpenter and S. Brim. Middleboxes: Taxonomy and Issues. RFC 3234 (Informational), February 2002. URL http://www.ietf.org/rfc/rfc3234.txt. 10, 120

Carrier Ethernet News. Upgrading to Bigger Backhaul Bandwidth for DSLAMs. http://www.carrierethernetnews.com/articles/319137/upgrading-to-bigger-backhaul-bandwidth-for-dslams/, 2011. 93

M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking Control of the Enterprise. *SIGCOMM Comput. Commun. Rev.*, 37(4):1–12, August 2007. 32

Cbench. Cbench: controller benchmark. http://docs.projectfloodlight.org/display/floodlightcontroller/Cbench, 2010, Online; accessed February 2014. 55, 63

Y. Chae and E. Zegura. CANEs: An execution environment for composable services. In *Proceedings of the 2002 DARPA Active Networks Conference and Exposition*, DANCE '02, pages 255–. IEEE Computer Society, 2002. 22

Y. Cheng, J. Chu, S. Radhakrishnan, and A. Jain. TCP Fast Open. Internet-Draft draft-ietf-tcpm-fastopen-05.txt, IETF, October 2013. 67

S. Cheshire and M. Krochmal. Multicast DNS. RFC 6762 (Proposed Standard), February 2013a. URL http://www.ietf.org/rfc/rfc6762.txt. 121

S. Cheshire and M. Krochmal. DNS-Based Service Discovery. RFC 6763 (Proposed Standard), February 2013b. URL http://www.ietf.org/rfc/rfc6763.txt. 121, 125

M. Chetty, J. Y. Sung, and R. E. Grinter. How Smart Homes Learn: The Evolution of the Networked Home and Household. In *UbiComp 2007: Ubiquitous Computing*, volume 4717 of *Lecture Notes in Computer Science*, pages 127–144. Springer Berlin Heidelberg, 2007. 74

M. Chetty, R. Banks, R. Harper, T. Regan, A. Sellen, C. Gkantsidis, T. Karagiannis, and P. Key. Who's hogging the bandwidth: the consequences of revealing the invisible in the home. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 659–668, Atlanta, Georgia, USA, 2010. ACM. 75, 97

Cisco. Express Forwarding Overview. http://www.cisco.com/en/US/docs/ios/12_2/switch/configuration/guide/xcfcef.html, May 2005, Online; accessed February 2014. 16

Cisco. Cisco Discovery Protocol Version 2. http://www.cisco.com/en/US/docs/ios-xml/ios/cdp/configuration/15-mt/nm-cdp-discover.html, November 2012a, Online; accessed February 2014. 18

Cisco. Cisco ASR 9000 Series Ethernet Line Cards. http://www.cisco.com/en/US/prod/collateral/routers/ps9853/data_sheet_c78-501338.html, 2012b. 96

Cisco. Per-VLAN Spanning Tree (PVST). http://www.cisco.com/en/US/tech/tk389/tk621/tk846/tsd_technology_support_sub-protocol_home.html, 2012c, Online; accessed February 2014. 17

Cisco. Per-VLAN Spanning Tree Plus(PVST+). http://www.cisco.com/en/US/tech/tk389/tk621/tk847/tsd_technology_support_sub-protocol_home.html, 2012d, Online; accessed February 2014. 17

D. Clark. The design philosophy of the DARPA internet protocols. *SIGCOMM Comput. Commun. Rev.*, 18(4):106–114, August 1988. 4

D. Cohen. Specifications for the Network Voice Protocol (NVP). RFC 741, November 1977. URL http://www.ietf.org/rfc/rfc741.txt. 4

Computer emergency response team. Vulnerability Note VU800113: Multiple DNS implementations vulnerable to cache poisoning. http://www.kb.cert.org/vuls/id/800113, July 2008, Online; accessed February 2014. 118

R. D. Corin, M. Gerola, R. Riggio, F. De Pellegrini, and E. Salvadori. VeRTIGO: Network virtualization and beyond. In *Proceedings of the European Workshop on Software Defined Networking (EWSDN), 2012*, EWSDN'12, pages 24–29, 2012. 32

G. A. Covington, G. Gibb, J. W. Lockwood, and N. Mckeown. A packet generator on the NetFPGA platform. In *Proceedings of the 17th IEEE Symposium on Field Programmable Custom Computing Machines*, FCCM '09, pages 235 –238, April 2009. 42

A. Crabtree, T. Rodden, T. Hemmings, and S. Benford. Finding a Place for UbiComp in the Home. In *UbiComp 2003: Ubiquitous Computing*, volume 2864 of *Lecture Notes in Computer Science*, pages 208–226. Springer Berlin Heidelberg, 2003. 74, 77

S. Crosby, S. Rooney, R. Isaacs, and H. Bos. A perspective on how ATM lost control. *SIGCOMM Comput. Commun. Rev.*, 32(5):25–28, November 2002. 25

CVNI. Cisco Visual Networking Index: Forecast and Methodology, 2011–2016. *CISCO White paper*, 2012a. 6

CVNI. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2011–2016 . *CISCO White Paper*, 2012b. 6

S. da Silva, Y. Yemini, and D. Florissi. The NetScript active network system. *Selected Areas in Communications, IEEE Journal on*, 19(3):538–551, 2001. 22

DARPA. Active Networks. http://www.sds.lcs.mit.edu/darpa-activenet/, April 1997, Online; accessed February 2014. 20

DCAN project. Devolved Control of ATM Networks. http://www.cl.cam.ac.uk/research/srg/netos/old-projects/dcan/, 2000, Online; accessed February 2014. 23

J. Dean and L. A. Barroso. The tail at scale. *Commun. ACM*, 56(2):74–80, February 2013. 106

D. S. Decasper, B. Plattner, G. M. Parulkar, Sumi C., J. D. DeHart, and T. Wolf. A scalable high-performance active network node. *Network, IEEE*, 13(1):8–19, 1999. 22

S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 1883 (Proposed Standard), December 1995. URL http://www.ietf.org/rfc/rfc1883.txt. 117

E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959. 18

R. Dingledine and N. Mathewson. Design of a blocking-resistant anonymity system. *The Tor Project, Technical Report*, 11:15–16, 2006. 103, 124

S. Dipjyoti. Mul OpenFlow controller. http://sourceforge.net/projects/mul/, 2013, Online; accessed February 2014. 31

M. Dischinger, A. Haeberlen, K. P. Gummadi, and S. Saroiu. Characterizing Residential Broadband Networks. In *Proceedings of the 7th ACM SIGCOMM Conference on*

Internet Measurement, IMC '07, pages 43–56, San Diego, California, USA, 2007. ACM. 93

A. Doria, F. Hellstrand, K. Sundell, and T. Worster. General Switch Management Protocol (GSMP) V3. RFC 3292 (Proposed Standard), June 2002. URL http://www.ietf.org/rfc/rfc3292.txt. 23

R. Droms. Dynamic Host Configuration Protocol. RFC 2131, IETF, March 1997. 82

D. Drutskoy, E. Keller, and J. Rexford. Scalable Network Virtualization in Software-Defined Networks. *Internet Computing, IEEE*, 17(2):20–27, 2013. 32

M. P. du Rausas, J. Manyika, E. Hazan, J. Bughin, M. Chui, and R. Said. Internet matters: The Net's sweeping impact on growth, jobs, and prosperity. http://www.mckinsey.com/insights/mgi/research/technology_and_innovation/internet_matters, May 2011. 5

N. Dukkipati and N. McKeown. Why Flow-completion Time is the Right Metric for Congestion Control. *SIGCOMM Comput. Commun. Rev.*, 36(1):59–62, January 2006. 70

N. Dukkipati, T. Refice, Y. Cheng, J. Chu, T. Herbert, A. Agarwal, A. Jain, and N. Sutin. An argument for increasing tcp's initial congestion window. *SIGCOMM Comput. Commun. Rev.*, 40(3):26–33, June 2010. 71

D. Eastlake 3rd. DNS Request and Transaction Signatures (SIG(0)s). RFC 2931 (Proposed Standard), September 2000. URL http://www.ietf.org/rfc/rfc2931.txt. 120

Endace. DAG Cards. http://www.endace.com/endace-dag-high-speed-packet-capture-cards.html, 2012, Online; accessed February 2014. 42

D. Erickson, B. Heller, S. Yang, J. Chu, J. Ellithorpe, S. Whyte, S. Stuart, N. McKeown, G. Parulkar, and M. Rosenblum. Optimizing a virtualized data center. *SIGCOMM Comput. Commun. Rev.*, 41(4):478–479, August 2011. 36

G. Even, M. Medina, G. Schaffrath, and S. Schmid. Competitive and Deterministic Embeddings of Virtual Networks. In *Proceedings of the 13th International Conference on Distributed Computing and Networking*, ICDCN'12, pages 106–121, Hong Kong, China, 2012. Springer-Verlag. 35

A. Finamore, M. Mellia, M. M. Munafò, R. Torres, and S. G. Rao. YouTube everywhere: impact of device and infrastructure synergies on user experience. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, IMC '11, pages 345–360, Berlin, Germany, 2011. ACM. 5

A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6824 (Experimental), January 2013. URL http://www.ietf.org/rfc/rfc6824.txt. 115

N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker. Frenetic: a network programming language. *SIGPLAN Not.*, 46(9): 279–291, September 2011. 34

git annex. git-annex. http://git-annex.branchable.com, 2013, Online; accessed February 2014. 130

Global Environment for Network Innovations (GENI). GENI: exploring networks of the future. http://www.geni.net, 2013. 36

Google. OpenFlow @ Google. http://www.opennetsummit.org/archives/apr12/hoelzle-tue-openflow.pdf, 2013, Online; accessed February 2014. 28

G. Greenwald and E. MacAskill. NSA Prism program taps in to user data of Apple, Google and others. http://www.guardian.co.uk/world/2013/jun/06/us-tech-giants-nsa-data, June 2013, Online; accessed February 2014. 107

R. E. Grinter, W. K. Edwards, M. W. Newman, and N. Ducheneaut. The Work to Make a Home Network Work. In *Proceedings of the 9th Conference on European Conference on Computer Supported Cooperative Work*, ECSCW'05, pages 469–488, Paris, France, 2005. Springer-Verlag New York, Inc. 8, 74

N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: towards an operating system for networks. *SIGCOMM Comput. Commun. Rev.*, 38, July 2008. 31, 40, 63, 78

A. Guha, M. Reitblatt, and N. Foster. Machine-verified network controllers. *SIGPLAN Not.*, 48(6):483–494, June 2013. 34

D. Gupta, K. V. Vishwanath, M. McNett, A. Vahdat, K. Yocum, A. Snoeren, and G. M. Voelker. DieCast: Testing Distributed Systems with an Accurate Scale Model. *ACM Trans. Comput. Syst.*, 29(2):4:1–4:48, May 2011. 56, 61

P. Gupta and N. McKeown. Algorithms for packet classification. *Network, IEEE*, 15 (2):24–32, 2001. 52

R. W Hahn and S. Wallsten. The economics of net neutrality. *The Economists' Voice*, 3(6), 2006. 93

N. Handigol, S. Seetharaman, M. Flajslik, R. Johari, and N. McKeown. Aster* x: Load-balancing as a network primitive. In *Proceedings of the 9th GENI Engineering Conference (Plenary)*, 2010. 33

N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown. Reproducible network experiments using container-based emulation. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, CoNEXT '12, pages 253–264, Nice, France, 2012a. ACM. 36

N. Handigol, B. Heller, V. Jeyakumar, D. Maziéres, and N. McKeown. Where is the debugger for my software-defined network? In *Proceedings of the 1st workshop on Hot topics in software defined networks*, HotSDN '12, pages 55–60, Helsinki, Finland, 2012b. ACM. 35, 56

Nikhil Handigol, S.vasan Seetharaman, Mario Flajslik, N. McKeown, and Ramesh Johari. Plug-n-Serve: Load-balancing web traffic using OpenFlow. *ACM SIGCOMM Demo*, 2009. 33, 54

J. J. Hartman, P. A. Bigot, P. Bridges, B. Montz, R. Piltz, O. Spatscheck, T. A. Proebsting, L. L. Peterson, and A. Bavier. Joust: a platform for liquid software. *Computer*, 32(4):50–56, 1999. 22

T. Hastings, R. Herriot, R. deBry, S. Isaacson, and P. Powell. Internet Printing Protocol/1.1: Model and Semantics. RFC 2911 (Proposed Standard), September 2000. URL http://www.ietf.org/rfc/rfc2911.txt. 104

S. Hatonen, A. Nyrhinen, L. Eggert, S. Strowes, P. Sarolahti, and M. Kojo. An experimental study of home gateway characteristics. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, IMC '10, pages 260–266, Melbourne, Australia, 2010. ACM. 10

C. L. Hedrick. An introduction to IGRP. Technical report, The State University of New Jersey, Center for Computers and Information Services, Laboratory for Computer Science Research,, August 1991. 19

B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown. ElasticTree: saving energy in data center networks. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, NSDI'10, pages 17–17, San Jose, California, 2010. USENIX Association. 34

T. R. Henderson, S. Roy, S. Floyd, and G. F. Riley. ns-3 project goals. In *Proceedings of the 2006 workshop on ns-2: the IP network simulator*, WNS2 '06, Pisa, Italy, 2006. ACM. 57, 62

M. Hicks, P. Kakkar, J. T. Moore, C. A. Gunter, and S. Nettles. PLAN: a packet language for active networks. *SIGPLAN Not.*, 34(1):86–93, September 1998. 21

M. Hicks, J.T. Moore, D.S. Alexander, C.A. Gunter, and S.M. Nettles. PLANet: an active internetwork. In *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3 of *INFOCOM'99*, pages 1124–1133 vol.3, 1999. 21

M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda. Is It Still Possible to Extend TCP? In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '11, pages 181–194, Berlin, Germany, 2011. ACM. 10

C. Hopps. Analysis of an Equal-Cost Multi-Path Algorithm. RFC 2992 (Informational), November 2000. URL http://www.ietf.org/rfc/rfc2992.txt. 30, 35

C. Hornig. A Standard for the Transmission of IP Datagrams over Ethernet Networks. RFC 894 (INTERNET STANDARD), April 1984. URL http://www.ietf.org/rfc/rfc894.txt. 4

HP. ProVision™ASIC: Built for the future. http://www.hp.com/rnd/itmgrnews/built_for_future.htm, January 2012, Online; accessed February 2014. 14

T. Y. Huang, K. K. Yap, B. Dodson, M. S. Lam, and N. McKeown. PhoneNet: a phone-to-phone network for group communication within an administrative domain. In *Proceedings of the 2nd ACM SIGCOMM workshop on Networking, systems, and applications on mobile handhelds*, MobiHeld '10, pages 27–32, New Delhi, India, 2010. ACM. 36

IEEE. IEEE Std 802.1D-1998: Media Access Control (MAC) Bridges. http://standards.ieee.org/getieee802/download/802.1D-1998.pdf, 1998, Online; accessed February 2014. 17

IEEE. IEEE Std 802.1D-2004: Media Access Control (MAC) Bridges. http://standards.ieee.org/getieee802/download/802.1D-2004.pdf, 2004, Online; accessed February 2014. 17

IEEE. IEEE Std 802.1Q: Virtual Bridged Local Area Networks. http://www.dcs.gla.ac.uk/~lewis/teaching/802.1Q-2005.pdf, 2005, Online; accessed February 2014. 17

IEEE. IEEE 802.1 aq Shortest Path Bridging. http://www.ieee802.org/1/pages/802.1aq.html, 2006, Online; accessed February 2014. 17

IEEE. IEEE 802.1 ab Station and Media Access Control Connectivity Discovery. http://standards.ieee.org/getieee802/download/802.1AB-2009.pdf, September 2009, Online; accessed February 2014. 18

Intel. Ethernet Switching Components. http://ark.intel.com/products/family/134/Intel-Ethernet-Switching-Components, 2014, Online; accessed February 2014. 14

iodine. iodine (IP-over-DNS, IPv4 over DNS tunnel). http://code.kryo.se/iodine/, 2014, Online; accessed February 2014. 113

ISO/IEC. *ISO/IEC FCD 8878, ITU-T Rec. X.233 – Information technology – Open distributed processing – Use of UML for ODP system specifications*, 1997. ISO/IEC FCD , ITU-T Recommendation X.233. 7

ISO/IEC. *ISO/IEC FCD 8348, ITU-T Rec. X.213 – Information technology – Open distributed processing – Use of UML for ODP system specifications*, 2001. ISO/IEC FCD , ITU-T Recommendation X.213. 7

T. Issariyakul. *Introduction to network simulator NS2*. Springer Science+ Business Media, 2012. 56

ITU. The World in 2011: ICT facts and figures, 2011, http://www.itu.int/ITU-D/ict/facts/2011/material/ICTFactsFigures2011.pdf. 4

M. Jarschel, F. Lehrieder, Z. Magyari, and R. Pries. A Flexible OpenFlow-Controller Benchmark. In *Proceedings of the European Workshop on Software Defined Networking (EWSDN), 2012*, pages 48–53, Oct 2012. 41

L. Jose, M. Yu, and J. Rexford. Online Measurement of Large Traffic Aggregates on Commodity Switches. In *Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, Hot-ICE'11, pages 13–13, Boston, MA, 2011. USENIX Association. 52

JSON-RPC Working Group et al. Json-rpc 2.0 specification. http://www.jsonrpc.org/specification, 2012, Online; accessed February 2014. 113

Juniper. T-series core router architecture overview. http://www.slideshare.net/junipernetworks/t-series-core-router-architecture-review-whitepaper, 2012, Online; accessed February 2014. 16

S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. The nature of data center traffic: measurements & analysis. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, pages 202–208, Chicago, Illinois, USA, 2009. ACM. 68

R. Katz. The impact of broadband on the economy: Research to date and policy issues. *Trends in Telecommunication reform 2010*, 11:23–82, 2011. 5

A. Khurshid, W. Zhou, M. Caesar, and P. B. Godfrey. VeriFlow: verifying network-wide invariants in real time. In *Proceedings of the 1st workshop on Hot topics in software defined networks*, HotSDN '12, pages 49–54, Helsinki, Finland, 2012. ACM. 35

H. Kim, T. Benson, A. Akella, and N. Feamster. The evolution of network configuration: a tale of two campuses. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, IMC '11, pages 499–514, Berlin, Germany, 2011a. ACM. 8

W. Kim, A. Roopakalu, K. Y. Li, and V. S. Pai. Understanding and characterizing planetlab resource usage for federated network testbeds. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '11, pages 515–532, Berlin, Germany, 2011b. ACM. 56

M. Kobayashi, S. Seetharaman, G. Parulkar, G. Appenzeller, J. Little, J. van Reijendam, P. Weissmann, and N. McKeown. Maturing of OpenFlow and software-defined networking through deployments. *Computer Networks*, 61(0):151 – 175, 2014. Special issue on Future Internet Testbeds Äì Part I. 28

O. Kolkman and R. Gieben. DNSSEC Operational Practices. RFC 4641 (Informational), September 2006. URL http://www.ietf.org/rfc/rfc4641.txt. 121

T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker. Onix: a distributed control platform for large-scale production networks. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–6, Vancouver, BC, Canada, 2010. USENIX Association. 34, 137

V. Kotronis, X. Dimitropoulos, and B. Ager. Outsourcing the routing control logic: better internet routing based on SDN principles. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, HotNets-XI, pages 55–60, Redmond, Washington, 2012. ACM. 33

M. Krasnyansky, M. Yevmenkin, and F. Thiel. Universal TUN/TAP device driver. https://www.kernel.org/doc/Documentation/networking/tuntap.txt, 2002, Online; accessed February 2014. 123

B. Krishnamurthy and C. E. Wills. On the Leakage of Personally Identifiable Information via Online Social Networks. In *Proceedings of the 2Nd ACM Workshop on Online Social Networks*, WOSN '09, pages 7–12, Barcelona, Spain, 2009. ACM. 106

N. Kushman, S. Kandula, and D. Katabi. Can you hear me now?!: it must be BGP. *SIGCOMM Comput. Commun. Rev.*, 37(2):75–84, March 2007. 20

A. Kuzmanovic, A. Mondal, S. Floyd, and K. Ramakrishnan. Adding Explicit Congestion Notification (ECN) Capability to TCP's SYN/ACK Packets. RFC 5562 (Experimental), June 2009. URL http://www.ietf.org/rfc/rfc5562.txt. 9

T. Lang, P. Branch, and G. Armitage. A Synthetic Traffic Model for Quake3. In *Proceedings of the 2004 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, ACE '04, pages 233–238, Singapore, 2004. ACM. xi, 99

A. Leach. Greedy Sky admits: We crippled broadband with TOO MANY users, 2013. 94

P. J. Leach and D. C. Naik. A common internet file system (cifs/1.0) protocol. http://www.ubiqx.org/cifs/rfc-draft/draft-leach-cifs-v1-spec-02.txt, March 1997. 104

O. Levin and R. Even. High-Level Requirements for Tightly Coupled SIP Conferencing. RFC 4245 (Informational), November 2005. URL http://www.ietf.org/rfc/rfc4245.txt. 103

J. T. Lewis, R. Russell, F. Toomey, B. McGurk, S. Crosby, and I. M. Leslie. Practical connection admission control for ATM networks based on on-line measurements. *Computer Communications*, 21(17):1585 – 1596, 1998. 24

L. E. Li, Z. M. Mao, and J. Rexford. Toward Software-Defined Cellular Networks. In *Proceedings of the 2012 European Workshop on Software Defined Networking*, EWSDN '12, pages 7–12. IEEE Computer Society, 2012. 36

Y. Liu, K. P. Gummadi, B. Krishnamurthy, and A. Mislove. Analyzing Facebook Privacy Settings: User Expectations vs. Reality. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '11, pages 61–70, Berlin, Germany, 2011. ACM. 107

LXC. LXC Linux Containers. http://lxc.sourceforge.net, 2013. 36

A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft. Unikernels: Library Operating Systems for the Cloud. *SIGPLAN Not.*, 48(4):461–472, March 2013. 38, 57, 63

R. Mahajan, D. Wetherall, and T. Anderson. Understanding BGP misconfiguration. *SIGCOMM Comput. Commun. Rev.*, 32(4):3–16, August 2002. 8

G. Malkin. RIP Version 2. RFC 2453 (INTERNET STANDARD), November 1998. URL http://www.ietf.org/rfc/rfc2453.txt. 19

D. McCullagh. Dropbox confirms security glitch–no password required. http://cnet.co/kvVbpz, 2011, Online; accessed February 2014. 107

N. McKeown, M. Casado, and T. Edsall. CS244: Advanced Topics in Networking, Spring 2012. http://www.stanford.edu/class/cs244/2012/, 2012. 36

S. Merugu, S. Bhattacharjee, Y. Chae, M. Sanders, K. Calvert, and E. Zegura. Bowman and CANEs: Implementation of an active network. In *Proceedings of the annual Allerton conference on communication control and computing*, volume 37, pages 147–156. Citeseer, 1999. 22

Microsoft. Windows filtering platform. http://msdn.microsoft.com/en-us/library/windows/desktop/aa366510.aspx, 2012. 95

D. L. Mills and H. W. Braun. The NSFNET backbone network. *ACM SIGCOMM Computer Communication Review*, 17(5):191–196, 1987. 4

P.V. Mockapetris. Domain names - concepts and facilities. RFC 1034 (INTERNET STANDARD), November 1987. URL http://www.ietf.org/rfc/rfc1034.txt. 117

C. Monsanto, N. Foster, R. Harrison, and D. Walker. A compiler and run-time system for network programming languages. In *Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '12, pages 217–230, Philadelphia, PA, USA, 2012. ACM. 34

C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker. Composing software-defined networks. In *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation*, nsdi'13, pages 1–14, Lombard, IL, 2013. USENIX Association. 31, 34

A.B. Montz, D. Mosberger, S.W. O'Mally, L.L. Peterson, and T.A. Proebsting. Scout: a communications-oriented operating system. In *Proceedings of the 5th Workshop on Hot Topics in Operating Systems, 1995. (HotOS-V)*, pages 58–61, 1995. 22

R. Mortier, B. Bedwell, K. Glover, T. Lodge, T. Rodden, C. Rotsos, A. W. Moore, A. Koliousis, and J. Sventek. Supporting Novel Home Network Management Interfaces with Openflow and NOX. *SIGCOMM Comput. Commun. Rev.*, 41(4):464–465, August 2011. 78

J. Moy. OSPF Version 2. RFC 2328 (INTERNET STANDARD), April 1998. URL http://www.ietf.org/rfc/rfc2328.txt. 18

J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown. Implementing an OpenFlow switch on the NetFPGA platform. In *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '08, pages 1–9, San Jose, California, 2008. ACM. 43

netfilter.org project. libnetfilter conntrack project. http://www.netfilter.org/projects/libnetfilter_conntrack/index.html, 2010. 95

B. Nowicki. NFS: Network File System Protocol specification. RFC 1094 (Informational), March 1989. URL http://www.ietf.org/rfc/rfc1094.txt. 104

Ofelia. FP7 Ofelia. http://www.fp7-ofelia.eu/, 2013. 36, 56

OFLOPS. OFLOPS manual, 2011, http://www.openflow.org/wk/index.php/Oflops. 41

R. Olsson. *pktgen*, the linux packet generator. In *Linux Symposium*, volume 2, pages 11–24, July 2005. 42, 88

S. W. O'Malley and L. L. Peterson. A dynamic network architecture. *ACM Trans. Comput. Syst.*, 10(2):110–143, May 1992. 20

L. Ong and J. Yoakum. An Introduction to the Stream Control Transmission Protocol (SCTP). RFC 3286 (Informational), May 2002. URL http://www.ietf.org/rfc/rfc3286.txt. 115

Open Network Foundation. Software-Defined Networking: The New Norm for Networks. https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf, 2012, Online; accessed February 2014. 25

OpenFlow Consortium. OpenFlow reference implementation. http://www.openflow.org/wp/develop/, 2009a. 43

OpenFlow Consortium. OpenFlow Switch Specification (version 1.0.0). www.openflow.org/documents/openflow-spec-v1.0.0.pdf, December 2009b. 42, 45, 48

OpenVPN. OpenVPN. http://openvpn.net, 2014, Online; accessed February 2014. 103

D. Oran. OSI IS-IS Intra-domain Routing Protocol. RFC 1142 (Historic), February 1990. URL http://www.ietf.org/rfc/rfc1142.txt. 18

E. D. Osborne and A. Simha. *Traffic engineering with MPLS*. Cisco Press, 2002. 18

A. Pathak, M. Zhang, Y. C. Hu, R. Mahajan, and D. Maltz. Latency inflation with MPLS-based traffic engineering. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, IMC '11, pages 463–472, Berlin, Germany, 2011. ACM. 18

J. Pelkey and G. Riley. Distributed simulation with MPI in ns-3. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, SIMUTools '11, pages 410–414, Barcelona, Spain, 2011. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). 63

J. C. Perez. Facebook's Beacon More Intrusive Than Previously Thought. http://www.pcworld.com/article/140182/article.html, November 2007, Online; accessed February 2014. 106

R. Perlman. An algorithm for distributed computation of a spanning tree in an extended LAN. *SIGCOMM Comput. Commun. Rev.*, 15(4):44–53, September 1985. 17

L.L. Peterson and B.S. Davie. *Computer Networks: A Systems Approach*. The Morgan Kaufmann Series in Networking. Elsevier Science, 2011. ISBN 9780123850607. URL http://books.google.co.uk/books?id=BvaFreun1W8C. 5

J. Pettit, J. Gross, B. Pfaff, M. Casado, and S. Crosby. Virtualizing the Network Forwarding Plane. In *Proceedings of the 2nd Workshop on Data Center - Converged and Virtual Ethernet Switching*, 2010. 43, 44, 78

PlanetLab. PlanetLab: An open platform for developing, deploying and accessing planetary-scale services, 2007. URL http://www.planet-lab.org. 56

E. S. Poole, M. Chetty, R. E. Grinter, and W. K. Edwards. More Than Meets the Eye: Transforming the User Experience of Home Network Management. In *Proceedings of the 7th ACM Conference on Designing Interactive Systems*, DIS '08, pages 455–464, Cape Town, South Africa, 2008. ACM. 74

P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu. A security enforcement kernel for OpenFlow networks. In *Proceedings of the 1st workshop on Hot topics in software defined networks*, HotSDN '12, pages 121–126, Helsinki, Finland, 2012. ACM. 32

J. Postel. Internet Protocol. RFC 791 (INTERNET STANDARD), September 1981a. URL http://www.ietf.org/rfc/rfc791.txt. 4, 10

J. Postel. Transmission Control Protocol. RFC 793 (INTERNET STANDARD), September 1981b. URL http://www.ietf.org/rfc/rfc793.txt. 4

J. A. Pouwelse, P. Garbacki, D. Epema, and H. J. Sips. A measurement study of the BitTorrent peer-to-peer file-sharing system. Technical Report PDS-2004-003, Delft University of Technology, The Netherlands, 2004. 5

Pox. About Pox. http://www.noxrepo.org/pox/about-pox/, 2012, Online; accessed February 2014. 31

I. Pratt. Pratt's Test TCP. http://www.cl.cam.ac.uk/research/srg/netos/netx/index.html, 2002. 59

Privoxy. Privoxy - Home Page. http://privoxy.org, 2013, Online; accessed February 2014. 103, 123

Z. Puljiz and M. Mikuc. IMUNES Based Distributed Network Emulator. In *International Conference on Software in Telecommunications and Computer Networks*, SoftCOM'06, Sept 2006. 56

Quagga. Quagga Routing Suite. http://www.quagga.net, 2014. 33

A. H. Rasti and R. Rejaie. Understanding peer-level performance in BitTorrent: A measurement study. In *Proceedings of the 16th International Conference on Computer Communications and Networks, 2007*, ICCCN'07, pages 109–114, 2007. 5

D. Recordon and D. Reed. OpenID 2.0: a platform for user-centric identity management. In *Proceedings of the 2nd ACM workshop on Digital identity management*, DIM '06, pages 11–16, Alexandria, Virginia, USA, 2006. ACM. 121

M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker. Abstractions for network update. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '12, pages 323–334, Helsinki, Finland, 2012. ACM. 34

Y. Rekhter. BGP Protocol Analysis. RFC 1265 (Informational), October 1991. URL http://www.ietf.org/rfc/rfc1265.txt. 19

E. Rescorla. Diffie-Hellman Key Agreement Method. RFC 2631 (Proposed Standard), June 1999. URL http://www.ietf.org/rfc/rfc2631.txt. 120

Jupiter Research and Akamai. RETAIL WEB SITE PERFORMANCE, Consumer Reaction to a Poor Online Shopping Experience. http://www.akamai.com/dl/reports/Site_Abandonment_Final_Report.pdf, June 2006, Online; accessed February 2014. 5

Return Infinity. BareMetalOS. http://www.returninfinity.com/baremetal.html, 2013. 58

T. Richardson and J. Levine. The Remote Framebuffer Protocol. RFC 6143 (Informational), March 2011. URL http://www.ietf.org/rfc/rfc6143.txt. 104

T. Rodden, A. Crabtree, T. Hemmings, B. Koleva, J. Humble, K. P. Akesson, and P. Hansson. Between the Dazzle of a New Building and Its Eventual Corpse: Assembling the Ubiquitous Home. In *Proceedings of the 5th Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques*, DIS '04, pages 71–80, Cambridge, MA, USA, 2004. ACM. 74

T. Rodden, A. Crabtree, T. Hemmings, B. Koleva, J. Humble, K. P. Akesson, and P. Hansson. Assembling Connected Cooperative Residential Domains. In *The Disappearing Computer*, volume 4500 of *Lecture Notes in Computer Science*, pages 120–142. Springer Berlin Heidelberg, 2007. 74

S. Rooney. The Hollowman: an innovative ATM control architecture. In *Proceedings of the 5th IFIP/IEEE international symposium on Integrated network management*, pages 369–380, San Diego, California, USA, 1997. Chapman & Hall, Ltd. 23

S. Rooney, J. E. van der Merwe, S. A. Crosby, and I. M. Leslie. The Tempest: a framework for safe, resource assured, programmable networks. *Communications Magazine, IEEE*, 36(10):42–53, 1998. 23

E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol Label Switching Architecture. RFC 3031 (Proposed Standard), January 2001. URL http://www.ietf.org/rfc/rfc3031.txt. 18

J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs). RFC 3489 (Proposed Standard), March 2003. URL http://www.ietf.org/rfc/rfc3489.txt. 127

C. E. Rothenberg, M. R. Nascimento, M. R. Salvador, C. N. A. Corrêa, S. Cunha de Lucena, and R. Raszuk. Revisiting routing control platforms with the eyes and muscles of software-defined networking. In *Proceedings of the 1st workshop on Hot topics in software defined networks*, HotSDN '12, pages 13–18, Helsinki, Finland, 2012. ACM. 33

C. Rotsos, H. Howard, D. Sheets, R. Mortier, A. Madhavapeddy, Amir Chaudhry, and J. Crowcroft. Lost in the edge: Finding your way with dnssec signposts. In *Proceedings of the 3rd USENIX Workshop on Free and Open Communications on the Internet*, Washington, D.C., 2013. USENIX. 102

N. Sarrar, S. Uhlig, A. Feldmann, R. Sherwood, and X. Huang. Leveraging Zipf's law for traffic offloading. *SIGCOMM Comput. Commun. Rev.*, 42(1):16–22, January 2012. 33

B. Schwartz, A. W. Jackson, W. T. Strayer, W. Zhou, R. D. Rockwell, and C. Partridge. Smart packets: applying active networks to network management. *ACM Trans. Comput. Syst.*, 18(1):67–88, February 2000. 21

A. Shaikh and A. Greenberg. Experience in Black-box OSPF Measurement. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, IMW '01, pages 113–125, San Francisco, California, USA, 2001. ACM. 48

E. Shehan and W. K. Edwards. Home Networking and HCI: What Hath God Wrought? In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 547–556, San Jose, California, USA, 2007. ACM. 74

S. Shepler, M. Eisler, and D. Noveck. Network File System (NFS) Version 4 Minor Version 1 Protocol. RFC 5661 (Proposed Standard), January 2010. URL http://www.ietf.org/rfc/rfc5661.txt. 130

R. Sherwood, M. Chan, A. Covington, G. Gibb, M. Flajslik, N. Handigol, T. Y. Huang, P. Kazemian, M. Kobayashi, J. Naous, S. Seetharaman, D. Underhill, T. Yabe, K. K. Yap, Y. Yiakoumis, H. Zeng, G. Appenzeller, R. Johari, N. McKeown, and G. Parulkar. Carving research slices out of your production networks with OpenFlow. *SIGCOMM Comput. Commun. Rev.*, 40:129–130, January 2010a. 32

R. Sherwood, G. Gibb, K. K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. Can the Production Network Be the Testbed? In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 1–6, Vancouver, BC, Canada, 2010b. USENIX Association. 95, 137

K.-Y. Siu and R. Jain. A brief overview of ATM: protocol layers, LAN emulation, and traffic management. *SIGCOMM Comput. Commun. Rev.*, 25(2):6–20, April 1995. 7

J. Y. Sung, L. Guo, R. E. Grinter, and H. I. Christensen. "My Roomba Is Rambo": Intimate Home Appliances. In *UbiComp 2007: Ubiquitous Computing*, volume 4717 of *Lecture Notes in Computer Science*, pages 145–162. Springer Berlin Heidelberg, 2007. 74

J. Sventek, A. Koliousis, O. Sharma, N. Dulay, D. Pediaditakis, M. Sloman, T. Rodden, T. Lodge, B. Bedwell, K. Glover, and R. Mortier. An information plane architecture supporting home network management. In *Proceedings of the IFIP/IEEE 2011 nternational Symposium on Integrated Network Management (IM)*, IM'11, pages 1–8, May 2011. 78, 79

Big Switch. Floodlight OpenFlow Controller. http://www.projectfloodlight.org/floodlight/, 2013. 31, 40

B. Teitelbaum and S. Shalunov. Why Premium IP Service Has Not Deployed (and Probably Never Will) . Technical report, Internet2, May 2002. URL http://qos.internet2.edu/wg/documents-informational/20020503-premium-problems-non-architectural.html. 9

P. Tolmie, A. Crabtree, T. Rodden, C. Greenhalgh, and S. Benford. Making the home network at home: Digital housekeeping. In *ECSCW 2007*, pages 331–350. Springer London, Limerick, Ireland, 2007. 74, 76

A. Tootoonchian, M. Ghobadi, and Y. Ganjali. OpenTM: Traffic Matrix Estimator for OpenFlow Networks. In *Proceedings of the 11th International Conference on Passive and Active Measurement*, PAM'10, pages 201–210, Zurich, Switzerland, 2010. Springer-Verlag. 52

J. Touch, D. Black, and Y. Wang. Problem and Applicability Statement for Better-Than-Nothing Security (BTNS). RFC 5387 (Informational), November 2008. URL http://www.ietf.org/rfc/rfc5387.txt. 105

Trema. Trema: Full-Stack OpenFlow Framework in Ruby and C. http://trema.github.io/trema/, 2011, Online; accessed February 2014. 31

A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker. Scalability and Accuracy in a Large-scale Network Emulator. *SIGOPS Oper. Syst. Rev.*, 36(SI):271–284, December 2002. 55

J. E. van der Merwe. Open service support for ATM. Technical Report UCAM-CL-TR-450, University of Cambridge, Computer Laboratory, November 1998. ix, 23

A. Varga and R. Hornig. An overview of the OMNeT++ simulation environment. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, Simutools '08, pages 60:1–60:10, Marseille, France, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). 56

Virgin Media. Cable traffic management policy. http://help.virginmedia.com/system/selfservice.controller?CMD=VIEW_ARTICLE&ARTICLE_ID=2781, 2014. 94

A. Voellmy, A. Agarwal, and P. Hudak. Nettle: Functional Reactive Programming for OpenFlow Networks. Technical Report YALEU/DCS/RR-1431, Yale University, July 2010. 31

A. Voellmy, H. Kim, and N. Feamster. Procera: a language for high-level reactive network control. In *Proceedings of the 1st workshop on Hot topics in software defined networks*, HotSDN '12, pages 43–48, Helsinki, Finland, 2012. ACM. 34

VTun. VTun - virtual Tunnels over TCP/IP networks. http://vtun.sourceforge.net, 2012, Online; accessed February 2014. 125

R. Wang, Dana Butnariu, and J. Rexford. OpenFlow-based server load balancing gone wild. In *Proceedings of the 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services*, Hot-ICE'11, pages 12–12, Boston, MA, 2011. USENIX Association. 33

D. Watson, F. Jahanian, and C. Labovitz. Experiences with monitoring OSPF on a regional service provider network. In *Proceedings of the 23rd International Conference on Distributed Computing Systems*, ICDCS '03, pages 204–. IEEE Computer Society, 2003. 20

S. Weiler. DNSSEC Lookaside Validation (DLV). RFC 5074 (Informational), November 2007. URL http://www.ietf.org/rfc/rfc5074.txt. 121

P. Weissmann and S. Seetharaman. How mature is OpenFlow to be introduced in production networks, 2011. URL http://changeofelia.info.ucl.ac.be/pmwiki/uploads/SummerSchool/Program/session_004.pdf. 39

D. J. Wetherall, J. V. Guttag, and D.L. Tennenhouse. ANTS: a toolkit for building and dynamically deploying network protocols. In *Open Architectures and Network Programming, 1998 IEEE*, pages 117–129, 1998. 21

Mike P. Wittie, V. Pejovic, L. Deek, K. C. Almeroth, and B. Y. Zhao. Exploiting locality of interest in online social networks. In *Proceedings of the 6th International Conference*, Co-NEXT '10, pages 25:1–25:12, Philadelphia, Pennsylvania, 2010. ACM. 106

A. Wundsam, D. Levin, S. Seetharaman, and A. Feldmann. OFRewind: enabling record and replay troubleshooting for networks. In *Proceedings of the 2011 USENIX conference on USENIX annual technical conference*, USENIXATC'11, pages 29–29, Portland, OR, 2011. USENIX Association. 35

Xen Project. XAPI. http://www.xenproject.org/developers/teams/xapi.html, 2013, Online; accessed February 2014. 61

Xilinx Inc. Virtex-5 Family Overview. http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf, 2009, Online; accessed February 2014. 15

K. K. Yap, M. Kobayashi, D. Underhill, S. Seetharaman, P. Kazemian, and N. McKeown. The Stanford OpenRoads deployment. In *Proceedings of the 4th ACM International Workshop on Experimental Evaluation and Characterization*, WINTECH '09, pages 59–66, Beijing, China, 2009. ACM. 36, 54

K. K. Yap, R. Sherwood, M. Kobayashi, T. Y. Huang, M. Chan, N. Handigol, N. McKeown, and G. Parulkar. Blueprint for introducing innovation into wireless mobile networks. In *Proceedings of the 2nd ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures*, VISA '10, pages 25–32, New Delhi, India, 2010. ACM. 36

Y. Yiakoumis, K.-K. Yap, S. Katti, G. Parulkar, and N. McKeown. Slicing home networks. In *Proceedings of the 2nd ACM SIGCOMM workshop on Home networks*, HomeNets '11, pages 1–6, Toronto, Ontario, Canada, 2011. ACM. 35

Y. Yiakoumis, S. Katti, T. Y. Huang, N. McKeown, K. K. Yap, and R. Johari. Putting home users in charge of their network. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, pages 1114–1119, Pittsburgh, Pennsylvania, 2012. ACM. 35

T. Ylonen and C. Lonvick. The Secure Shell (SSH) Transport Layer Protocol. RFC 4253 (Proposed Standard), January 2006. URL http://www.ietf.org/rfc/rfc4253.txt. 104

M. Yu, J. Rexford, M. J. Freedman, and J. Wang. Scalable flow-based networking with DIFANE. *SIGCOMM Comput. Commun. Rev.*, 41(4):–, August 2010. 33