Scalable Software Defined Networking

Charalampos Rotsos

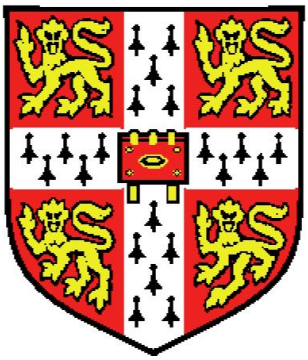
Computer Laboratory
University of Cambridge

A thesis submitted for the degree of
Doctor of Philosophy

Yet to be decided

Abstract

Scalable Software Defined Networking



Charalampos Rotsos

Computer Laboratory

University of Cambridge

A thesis submitted for the degree of

Doctor of Philosophy

Yet to be decided

Contents

List of Figures

Todo list

24, add GSMP protocol reference

30, this is a simplified description for the protocol model. Different protocol version extend some aspects of the model

32, this is shit

77, add reference to Mazurek et al. [2010] for access control requirements for the house.

List of Publications

As part of my PhD work the following work was published by me:

- **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 '10).

- R. Mortier, B. Bedwell, K. Glover, T. Lodge, T. Rodden, **C. Rotsos**, et al. *DEMO: Supporting novel home network management interfaces with openflow and NOX*. Presented at the SIGCOMM '11: Proceedings of the ACM SIGCOMM 2011 conference.

- R. Mortier, T. Rodden, T. Lodge, D. McAuley, **C. Rotsos**, A. W. Moore, A. Koliousis, J. Sventek, *Control and understanding: Owning your home network*, In Proceedings of the Fourth 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'12).

- A. Chaudhry, A. Madhavapeddy, **C. Rotsos**, R. Mortier, A. Aucinas, J. Crowcroft, et al. *DEMO: Signposts: end-to-end networking in a world of middleboxes*. Presented at the SIGCOMM '12: Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication.

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

- 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).

Chapter 1

Introduction

Computer networks have become the predominant interconnection fabric of our modern-era. Nonetheless, the widespread usage of network technologies has highlighted several scalability problems in network functionality, primarily deriving from the design of network protocols and architectures. Addressing these limitation through protocol redesign is a challenging process, which requires significant time and effort for deployment.

This dissertation is concerned with performance scalability problems in modern networks. The thesis of this dissertation is that control plane redesign can mitigate network bottleneck, introduced by protocol design, while ensuring backwards compatibility and high performance. We argue that existing control plane mechanisms remain inflexible and their "single abstraction fits all" approach fails to fulfil the unique requirements for a set of novel deployment environments, e.g. data centers and home networks. We advocate that modern networks require control redesign, *specialised* to the requirement and opportunities of the environment. The recent introduction of SDN in network devices provides a sufficient and readily-available enabler for our 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 concept 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 an heterogeneous computer network of significant size, was *ARPANET* Beranek [1981]. ARPANET improved the resilience and performance in comparison to circuit-switched networks, while its simplified design allowed easy integration of existing computing systems. ARPANET 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, specifically 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 finally evolved in today's Internet. As part of this transition, the research community developed the standards for the Internet protocol suite Clark [1988], the lower and medium layers of the network stack.

Computer networks, conceived only 20 years after the implementation of a programmable computer, have gained a significant position in our society since the ARPANET years, guided by the digital revolution. Their communication abstraction currently replaces a number of traditional communication systems, while providing 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], while Internet resources are estimated to generate 3.4% of global GDP du Rausas et al. [2011]. In developed economies an average users is connected to the Internet in parallel over multiple devices (e.g. lap-

top, smartphone, home entertainment system) and a large portion of daily activities relies on network technologies (e.g. e-banking, e-gov, online social networking). Furthermore, broadband connectivity increase in developing economies exhibits a positive correlation with GDP growth (10% increase in broadband connectivity improves GDP by approximately 1% Katz [2011]). The growth in network adoption introduces novel performance requirements for network technologies and protocols.

1.1.2 The Lernaean Hydra of Network Performance

According to Wikipedia [2014]:

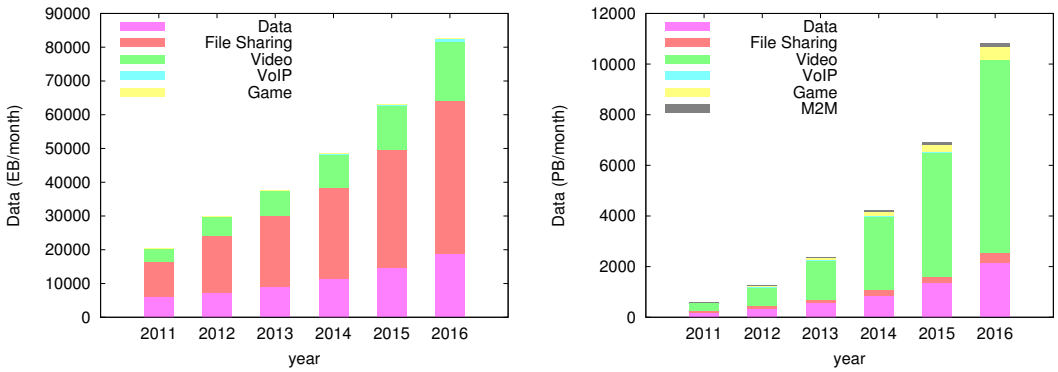
> **Network performance** refers to measures of service quality of a telecommunications product as seen by the customer.

Based on this definition, network performance is highly subjective, defined by the network environment, the application requirements and the user. The increased adoption of network technologies creates an equal augmentation in network use-cases and, consequently, the definition of network performance becomes complex and domain and application-specific. In addition, network performance metrics are not limited to pure network packet processing capabilities and consider other, potentially subjective, aspects, like usability. We elaborate on the complexity and multiple dimensions of performance discussing two example aspects: application resource requirements and network heterogeneity.

Simplicity and multi-device availability of the network abstraction have given rise to the development of a wide range of applications, with global audiences 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. Peer to peer applications have no performance requirement but aim to maximize network throughput, while VoIP application have strict latency requirements. As a result, applications face difficulties to share network resources during congestion. Network operators overcome the sharing problem by over-provisioning bandwidth resources, but physical limitations reduce the effectiveness of the approach.

Application	throughput (Mbps)	latency (sec)	jitter (sec)	flow number (median)
web Butkiewicz et al. [2011]; JupiterResearch 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 a set of popular traffic classes.



(a) global Internet traffic trend

(b) global mobile Internet traffic trend

Figure 1.1: Cisco Visual Network Index reports on global network traffic per application for Internet (Figure 1.1(b)) and for Mobile Internet (Figure 1.1(a)).

Application performance exhibits similar complexity in a macroscopic level. Figure 1.1 presents estimated traffic volumes for six popular network application classes between the years 2011-2016 for mobile and fixed Internet, using data from the Cisco visualization index Cisco [2012e]; Mobile [2012]. Aggregate network traffic volumes exhibit an exponential increase and are expected to augment an order of magnitude for mobile Internet and four times for fixed Internet between the years 2011 and 2016. Traffic volume increase, is primarily driven by video transmission applications which increase their popularity over the file-sharing applications. The popularity churn is explained by the fact that the predominant file-sharing content of video is now available through high-availability CDN services, like Youtube. Nonetheless, as the two classes exhibit different characteristics with respect to bandwidth requirements, network providers are obliged to map this change in their architecture and service configuration. Application classes commonly exhibit such popularity dynamics.Network application popularity can vary over time but a minimum volume will exist in the network and shape ISP policies and architectures.

A significant evolution exists also 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 low design complexity. The network community has defined standards to implement the Ethernet abstraction over a wide range of medium types, like coper, fibber, 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 experience 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 the 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 In-

ternet over the recent years, but the protocol designs haven't been able to keep up with the change. As a result, the mismatch between protocol capabilities and requirements becomes evident, due to traffic rates and size increase.

An interesting observation on the evolution of network technologies, and especially of their control plane, is that abstraction simplicity is a prime 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 which where motivated by the design choices, like fixed cell size and circuit-based forwarding approach. Nonetheless, their high complexity faces performance scalability problems, with respect to the available computational resources, and the protocols became obsolete in favour of the TCP/IP protocol suite. For the rest of the section, we discuss in detail some example of network bottlenecks which are a consequence of the design of current network protocols.

Network 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 misconfigurations, prompted to a great extend by the counter-intuitive configuration interface. Kim et al. [2011] highlight limitations of current configuration interfaces by studying the configuration evolution of two medium-sized campus networks. Device configuration files exhibit a continuous size increase, containing frequently stale rules, 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 policy are merged by the device interface. The configuration bottleneck is equally important in environments with non-expert administrators, like the home network. Grinter and Edwards [2005] conclude that human understanding of the network abstraction is limited and the complexity of the control abstraction leads to under-optimized network configuration, especially in terms of security and performance.

Currently, network configuration relies to a great extent on the network administrator, who is responsible to manually transform 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 radical development of network technologies has limited effect on device interfaces, which remain low-level and counter-intuitive to administrators. As a result, network configuration remains still a process requiring and relying on the network administrator experience. Effectively, network management interfaces introduce a bottleneck on the "mechanism" translating the network policy into distributed device configurations, the network administrator.

Network management requires a technological "revolution", similar to the introduction of high-level programming language 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 though exhibits reduced effectiveness at high link-rates. High access speeds increase application responsiveness and shape equivalently end-user expectations. During high link utilisation though, network responsiveness and, consequently, user experience degrade severely, in the vicinity of queueing delays and packet losses. A number of mechanisms has been proposed to improve resource control, in multiple layers of the network stack (e.g. ECN Kuzmanovic et al. [2009], DiffServ Blake et al. [1998], RSVP over MPLS-TEAwduche et al. [2001]). These mechanisms, for different reasons, fail to provide a generic solution to the problem.

The ideal mechanism for effective resource control requires a-priori knowldge of resource requirement from end-hosts, in order to define optimal resource allocations.

Current network 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 losses and RTT variance, and adjusts accordingly the traffic rate. 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 effectively enforce resource control policy. Nonetheless, because the two mechanisms are weakly integrated, current resource control exhibits significant latency with respect to network access speeds. Effective network resource management requires smarter control planes, which augment the inputs to the decision algorithm and provide faster response.

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 redefined to a great extend overall protocol functionality, as they commonly introduce new assumptions in the protocol functionality and violate the Robustness principle of networks [1]. Honda et al. [2011] presents an Internet-wide robustness measurement and report that obscure but correct protocol behaviours turn out to be unsupported by a number of end-points, primarily due to the functionality of such devices.

An important 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 Eggert et al. [2008], but their generality is limited and context-specific.

[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 [1981]

1.2 Contributions

This dissertation contends that existing network architectures face a significant performance bottleneck in network functionality and provide limited support to the constantly evolving network requirements of end-hosts. Modern networks require flexible and specialized control approaches, which address complex functional requirements and unify network control across all devices of the network. Our study employs extensively the SDN control approach and the OpenFlow protocol. Nonetheless, the applicability of our contributions are not limited and can easily adapt to alternative programmable control frameworks. In detail, we claim the following specific contributions:

- **Control scalability evaluation**: We provide an in-depth study of the SDN control scalability. 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 interaction. SDNSIM is an experimentation platform, specialised for OpenFlow technologies, which allows a user to describe and either simulate or emulate an experimental setup. These two platforms establish a generic toolbox, providing flexible evaluation of SDN control applications. Using these tools, we provide an in-depth analysis of the control-plane performance and limitations for a wide range of off-the-self OpenFlow switch implementations. Additionally, using as input the OFLOPS switch performance models, we explore the impact of a hierarchical control scheme in a small-scale datacenter architecture.

- **Management scalability**: We present a control application which addresses the problem of network management and resource control in the context of home networks. 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 requirement 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 social input in the forwarding decision and improves both access control and QoS.

- **Connectivity scalability**: We revisit the problem of connectivity 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, 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 connectivity and controllable performance and security, across the network. In addition, 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 limitation of network control, we revisit control-plane mechanisms and related research efforts for control-plane scalability.

In Chapter 3 we study 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 device implementations and architectures, respectively. Using these tools, we conduct a measurement study to characterise the performance of baseline OpenFlow operation by SDN-enabled devices and the impact of distributed control architectures in the data plane performance. In Chapter 4, we study the problem of control plane management performance, focusing on the home network setting. Using existing user studies we draw the user network management requirements and redesign the control plane architecture of the home router. Our architecture provides a simpler, user-friendly control abstraction which matches the user requirement for control and information from their network. In addition, we provide an extensive evaluation of the architecture and provide evidence on the scalability and backwards compatibility of our architecture. In Chapter 5 we investigate application of the SDN paradigm in connectivity scalability. We present Signpost, a novel user-centric control plane architecture for end-hosts, providing Internet-wide device connectivity with user-controlled performance and security. We present the architec-

ture of the framework, its integration with existing connectivity-enabling mechanisms and we evaluate the performance and applicability of the framework. Finally, in Chapter 6 we draw conclusions from our study and suggest further work.

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, focusing on some of their deficiencies (Section 2.2). Furthermore, we present three important network control frameworks, namely Active Networks, Devolved Control of ATM Networks (DCAN) and Software Defined Networking (SDN), proposed by the research community to address some of the limitation of current control schemes (Section 2.2). Finally, we focus on the most recent control framework, SDN, and present some of its applications in modern network problems (Section 2.4).

2.1 Forwarding Devices

Packet networks have become a *'de-facto'* approach in communication networks. Their success relies, to a great extend, on the offered high medium utilization and low-cost design principle. Packet-switched networks can function over a large set of physical layer technologies and replace tradition connection-oriented mechanisms (e.g. telephony, TV broadcast). In this section we focus on Ethernet and IP networks, the predominant protocol implementations for packet networks, and provide a design overview of a hardware *Switch*, a typical forwarding device.

Network switches are the main mechanism to multiplex Ethernet links. Their

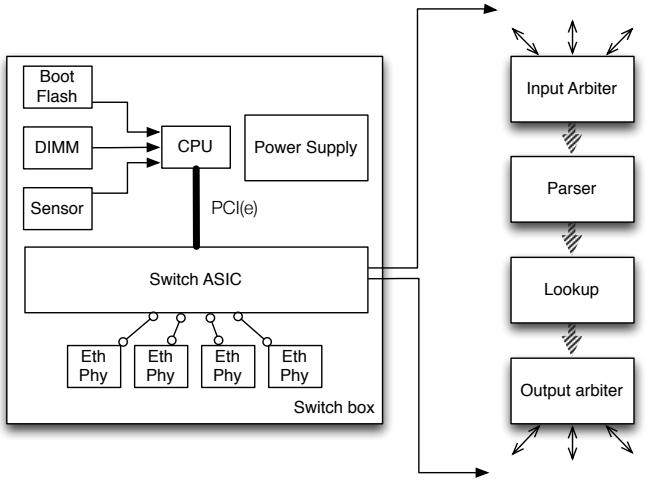


Figure 2.1: A generic architecture of a Switching device.

main functionality provides collision-free connectivity between network segments. Nonetheless, as the network complexity increases, switches augment their functional roles in a network and their hardware architecture is highly diverse. For example, unmanaged switches provide a low cost non-configurable interconnection device for small networks, supporting usually a few 1 gig ports, while distribution switches are used in large enterprise networks to interconnect aggregation switches, support multiple 10 or 40 Gig links and provide multiple mechanisms to inspect and control network traffic. In this section we will use the Top-of-Rach (ToR) switch type as an reference to elaborate on the architecture of modern switches. Such switches are used in datacenter networks to aggregate traffic between the edge and the distribution network layers. TOR switches commonly use a single Application-Specific Integrated Circuits (ASIC) silicon and can forward high-rate traffic non-blocking at line rate. As presented in Figure 2.1, a typical switch device consists of the following components:

- *management CPU*: Switch boxes contain a programmable CPU, which runs the control plane logic and provide configurability. Switch CPUs usually run a simple operating system and communicate with the switch ASICs in order to modify their functionality, based on the policy. Switch vendors usually use low-power CPUs, such as SoC PowerPC chips or even ARM and MIPS CPUs. The CPU can access runtime memory modules (Boot Flash and RAM), to store transient state, and persistent memory (e.g. SD Cards) through memory extension slots, to

store log files and configuration. In general, switch CPUs provide sufficient computation resources to run the switch control-plane functionality, but they cannot accommodate intensive processing tasks. The OS provide network services, like telnet and SSH, to provide a command line interface to network managers, while usually it also provides SNMP services to aggregate switch statistics.

- *Switch ASIC*: The switch ASIC Broadcom [2013]; HP [2012]; Intel implements in hardware the forwarding plane of the switch. The capabilities of an ASIC are variable, depending on the vendor and the cost, but define the performance limits of the device to handle data plane traffic. Implementation details of an ASIC silicon remain usually concealed, as they constitute an important asset for the producer. The design of an ASIC is an expensive process, as their design and testing period is long and requires significant human labour and material resources. As a result, switch silicon innovation has a high latency to reach the production environment. Furthermore, as in general purpose CPU design, the processing capacity of an ASIC is upper bound by the transistor density and size.

  TOR-type switches can handle line-rate non-blocking traffic for a number of ports, using a single ASIC silicon. An ASIC contains four important modules. The *input arbiter* module multiplexes and synchronizes packets from the Ethernet ports to the main processing pipeline of the silicon. The arbiter ensures non-preemptive packet processing, by bridging the rate mismatch between the silicon processing clock and the link rate. In the main processing pipeline the ASIC contains at least a *protocol parsing* module and an *memory lookup* module. The protocol parser extracts significant packet fields from network packets and the memory lookup module uses the packet fields to define the packet processing and forwarding operations. The lookup module relies on an interface with an external memory module in order to match protocol fields with forwarding policy. Memory modules have a trade-off between cost and access speed and between memory cost and memory management complexity [1]. Finally, the *output arbiter* module is responsible to apply modifications to the packet and forward it

[1] CAM/TCAM lookup: $<$ 10 clock cycles, SRAM: 2-3 nsec, DRAM: 20-35 nsec, http://people.ee.duke.edu/~sorin/prior-courses/ece152-spring2009/lectures/6.2.2-memory.pdf

to the appropriate output queue. Beyond the aforementioned modules, an ASIC may have additional processing modules in the main packet processing pipeline, to extend its functionality with operations like access control list (ACL) virtual network queues and flow statistics monitoring.

- *ASIC-CPU communication*: The switch CPU is connected with the ASIC over a PCI or PCI-express bus. The communication channel provides an elementary bi-directional channel: the CPU controls ASICs registers, to modify forwarding functionality, and the ASIC propagates information for exceptional states to the CPU. The channel provides sufficient bandwidth for basic control plane communication, but it is not designed to handle high rate information.

- *Memory*: Apart for the in-ASIC memory modules, modern 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 as a separate hardware circuit. The module is responsible to implement the physical layer and MAC layer protocol and propagates packets to the ASICs using a Media Independent Interface (MII). The phy module usually contains small packet buffers to reduce packet losses for bursty traffic.

The physical limitation for the control plane are defined by the ability of the management CPU to communicate with the ASIC. This communication can be further characterised by the speed of the switch to read and write ASIC registers and modify the ASIC memory modules. In addition, the computational requirement of the control plane logic can impact severely the control plane performance, if the management CPU processing capacity is limited.

TOR switches are simple switch devices, supporting the traffic requirements of the edge network and provide low to medium forwarding capacity. As we move closer to the core of the network, the processing capacity requirements, as well as, device complexity are increased. Multi-chassis switches and routers cannot contain the required

functionality in a single ASIC. Packet processing is distributed between multiple silicons, complex clos interconnection crossbars are used to provide non-blocking Terabit backplane capacity Juniper [2012] and the complexity of the control plane to modify forwarding policy is increased. In addition, providing fast address lookup requires multiple distributed buffers and cache coherent protocols to ensure state consistency Cisco [2005]. For these classes of devices, the control plane abstraction has a complex and slow translation to ASIC management operations and programming flexibility is restricted.

2.2 Forwarding Control in Production Networks

Dynamic network control has been an important functionality for computer network technologies. Forwarding devices require mechanism to automate forwarding logic update when network state changes in order to ensure end-to-end connectivity. Existing standardized approaches aim for *distributed*, *autonomous* and *resilient* network control and are influenced extensively by the end-to-end principle of the Internet. A switch or a router control functionality requires only local forwarding configuration. The device at run-time uses a number of well-defined distributed protocols to discover the global network state by state exchanges with other network devices. Using the global network state, the forwarding logic determines an optimal forwarding policy which maximizes network performance. Because the state is constructed collectively, current control plane can provide guarantees only for long-term resilience. If a link is disconnected, the control logic propagates this information across the network to update the network state in other devices, while if a device is rebooted, it can automatically rebuild the global network state using only its local configuration. In this section we present existing production-level protocols that enable distributed network control. We focus on the Data-Link and Network layer protocols as these layer define the hop-by-hop forwarding policy.

2.2.1 Data-Link Layer Control

Data-Link layer control protocols provide loop-detection, and VLAN and QoS automation. Loop detection is used to avoid packet loops in networks with redundant

connectivity, since the 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 graph of the network, with respect to a common root switch and controls a state machine per port, which disables packet forwarding when a link is not part of the spanning tree. Because the initial definition of the protocol faced significant convergence delay after a network change, IEEE introduce the Rapid Spanning Tree Protocol (RSTP), an evolve version of STP, in IEEE [2004]. In addition, a modified version of the protocol was defined to accommodate VLAN functionality, since VLAN tags can create multiple subgraphs over the network. The protocol runs individual STP algorithms for each subgraph and constructs individual spanning trees for each VLAN, thus reducing unnecessary port blocking. This functionality is provided by the IEEE Multiple Spanning Tree Protocol (MSTP) IEEE [2005], while Cisco has developed a series of proprietary protocols to address this problem Cisco [2012c,d]. Finally, IEEE has recently defined an advanced STP protocol that enables switches to discover and use in parallel redundant network links, in the IEEE 802.1aq standard IEEE [2006].

In terms of configuration automation, network vendors and standardization bodies have developed a number of protocols which disseminate device configuration to neighbouring nodes. Such protocols have high churn between vendors and each vendor so far has defined a custom standard. In this class of protocols we consider Link Layer Discovery Protocol (LLDP) IEEE [2009], supported by IEEE, 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 data link or network layer protocol which enables the creation of virtual-circuit over Ethernet networks. MPLS uses simple labels ID to forward packets and the protocol is used extensively in the distribution layer of large networks, to reduce forwarding table size. In order to automate the configuration of MPLS circuits, the MPLS community defined the LDP protocol Andrerson et al. [2007] to setup automatically labels across the network. In addition, IETF defined a mechanism to

translate RSVP resource requests to MPLS tunnel configuration Awduche et al. [2001] and the autobandwidth mechanism to enforce perfromance , since MPLS doesn't provide resource control, the IETF defined the *Autobandwidth* mechanism Osborne and Simha [2002]. Autobandwidth monitors tunnel bandwidth requirements and recalculates paths when resource requirements change. A study on a deployment of this technology in the MSN network though has highlighted that the allocation policy is under-optimal for significant periods of the network functionality Pathak et al. [2011].

2.2.2 Network Layer Control

In the network layer, modern network technologies exercise control using routing protocols. Routing protocols follow a similar approach to Data Link layer protocols and provide mechanisms to disseminate local configuration to the network. Routing protocols are classified in three categories: *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 to define the optimisation function of the routing system. Each router has 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 routing protocols employ a different approach to provide efficient global routing policies. 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. Distance Vector protocols use the Belman-Ford algorithm Bellman [1956] to achieve routing optimality. IETF developed the RIP protocol Malkin [1998] to implement Distance Vector routing,

while Cisco has developed the proprietary IGMP protocol Rutgers [1991]. Distance Vector protocols are less extensible than Link State routing protocols, but can be implemented with smaller computational and memory requirements.

Finally, Path Vector routing protocols evolve Distance Vector protocols to support inter-domain routing. In this class of routing protocols, adjacent routers advertise the AS path towards for a specific subnet. Path Vector routing permits ASes to abstract routing policy details, but provide sufficient information for the other ASes to define their forward policy. As a result, the BGP protocol doesn't provide an explicit routing protocol, but provides a sufficient framework for information exchange. The BGP protocol Rekhter [1991] is currently the predominant Path Vector routing implementation.

Due to their distributed nature, routing protocols provide long-term routing resilience, while their mathematical foundation is provably correct. Nonetheless, the control abstraction of these protocol is not a good fit for administrator to exercise dynamic control in 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 get bigger. In 2008 the global Internet was severely affected by a BGP misconfiguration from the Pakistani national ISP in an effort to control traffic from YouTube service Brown [2008]. In addition, routing inconsistencies during routing changes can impact significantly network performance. In Watson et al. [2003], the author evaluate the OSPF functionality in a regional ISP and report high churn in routing even during periods without significant network events, while the latency to converge after a network change is on average on the order of seconds. Similar results have been observer in BGP. In Kushman et al. [2007] the authors highlight a strong correlation of BGP instability and VoIP performance. Nonetheless, as the network link speeds increase, the impact of such update on the data plane of the network is increased, as the number of packet that will be affected by a routing instability is equally increased. Modern high speed networks require higher flexibility in the control abstraction and faster control plane responsiveness.

2.3 Programmable Network Control

Because of the limitation of current standardized network control frameworks, the research community over the years has proposed a number of approaches to evolve

network control. In this section we present the three most significant and complete approach to this problem, namely Active Networks, Devolved Control of ATM Networks (DCAN) and Software Defined Networking (SDN).

2.3.1 Active Networks

The network research community has highlighted the inflexibility to evolve network design, often termed *protocol ossification*, since mid 90's. O'Malley and Peterson [1992] challenged the generality of the OSI network model and suggested an evolved model which provided the ability to synthesize protocol parsers and incorporate variable numbers of layers in forwarding devices. Motivated by these observations, DARPA funded the *Active Networks* project DARPA [1997], in an effort to develop next generation network devices that can seamlessly integrate new protocol functionality.

Active networks modify the default packet processing mechanism and introduce the notion of *capsules*. Capsules are network packets that carry data, as well as, processing code. On each hop, the default packet processing logic is extended with the code contained in the capsule, thus redefining network functionality. This network processing approach provided user-driven protocols upgrades, without a requirement to modify devices. In addition, the development of novel programming languages, like Java, provided the building blocks to implement code distribution frameworks.

Research in active networks tried to define two primary building blocks for the Active Network implementation : 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]. The ANEP protocol was adopted by the ANTS and Sprocket capsule programming frameworks. Sprocket Schwartz et al. [2000] was developed by BBN technologies and defined a programming language, using a restricted set of C. The language avoided any insecure structures like pointers and provided native support for SNMP browsing. The sprocket compiler output MIPS assembly and the binary was executed using a MIPS VM on each network device. Sprocket did not persist any state on the switch, but allowed in capsule conde 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, due its intuitive se-

rialization functionality, while switch and capsule integration was modelled using Java interfaces. Capsule code was allowed to persist switch state and modify packet content. An interesting functionality of the ANTS framework was the code dissemination mechanism. An end-node deploys 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 Pensylvania active networks 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 multiple aspects of efficient capsule processing. University of Pensylvania proposed the SwitchWare Execution Environment (EE) Alexander et al. [1998a]. The architecture defined an OCaml module framework for capsule processing, with a strong focus on security, both on the switch, as well as, during capsule execution. SwitchWare used SANE Alexander et al. [1998b], a trustworthy operating system, to secure the functionality of the switch, and build on top of its security primitive higher level of trust. 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 networks.

The active networks group in University of Arizona proposed its custom approach to high performance active network forwarding. Their approach relies on the Scout OS Montz et al. [1995], a communication-oriented OS supporting efficient layered data processing. On top of Scout, the group developed the Liquid Software API Hartman et al. [1999]. The integration of Liquid Software API and Scout, aimed to establish a tight integration between the OS and the JVM and thus achieve improved capsule processing, using the JIT JAVA compiler.

The CANEs project Chae and Zegura [2002] from Georgia Tech Active Network group proposed a switch design, which increased flexibility in multi-protocol packet processing. In detail, 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 the Switch resources.

Researchers from the Columbia University, presented the NetScript switch abstraction da Silva et al. [2001]. NetScript is a new programming language optimized for flexible protocol processing definition and composition. The project defines three types of protocol composition: layered composition, composition through protocol bridging and end-to-end composition. Netscript provides developers the ability to build flexible extension in existing protocol processing.

Finally, a joint effort between the Network Laboratory of ETH and the University of St. Louis, developed a hybrid hardware and software approach for high performance capsule processing. The High Performance Active Network Node (ANN) switch architecture Decasper et al. [1999] proposes the introduction of an FPGA for each ATM interface which handles capsule execution. Using this infrastructure, the researchers where able to run the ANTS EE and develop an IPv4 and IPv6 protocol processor to run in Gigabit rates.

Active network research put forward a number of interesting insights on the controllability and evolvability problem in control plane functionality. Nonetheless, their complex and clean-slate approach, reduced Active Network applicability in production environments. In addition, Active Network functionality was highly benefited from the relatively low link rates of the time. CPU processing capability of the time, was able to accommodate the respective link rates. The rapid evolution of data rate in Ethernet interfaces to Gigabit speeds, through, makes impossible the development of programmable forwarding platforms with line rate performance in the current time.

2.3.2 Devolved Control of ATM Networks

add GSMP protocol reference Active Networks focused extensively on Ethernet technologies and strive to develop protocol evolvability. A similar approach for ATM was developer in the University of Cambridge for the DCAN project [2000] project, focusing though primarily on control evolvability and network virtualisation. DCAN tried primarily to simplify the control plane of ATM devices, in order to support network multi-functionality.

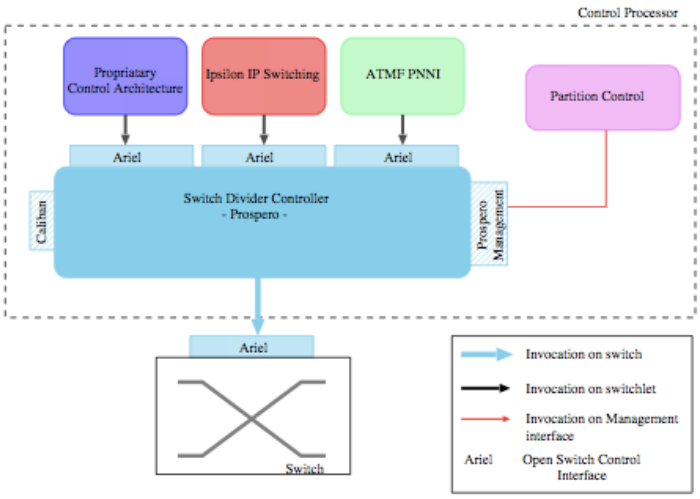


Figure 2.2: Tempest switch architecture van der Merwe [1998]

As part of the DCAN project, Rooney et al. [1998] present Tempest, a novel ATM switch architecture, which enabled clean separation between 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 to virtualise ATM switch resources into multiple virtual switches, called *switchlets*, through a resource control interface. A switchlet controls a subset of the switch ports, VCI and VPI mappings, packet buffer and bandwidth. In addition, Prospero used the ATM QoS principles to implement packet buffer and bandwidth virtualization. Fundamentally, Prospero is responsible to map the control interface, exposed to controllers, with the underlying switch functionality.

Ariel Switch Independent Control Interface Switchlets expose forwarding control through the Ariel Interface. Ariel Interface organises network control through

five control objects: *Configuration*, *Port*, *Context*, *Connections Statistics* and *Alarms*. Configuration provides details for the switch configuration, Port provides primitive controllability of ports (e.g. state, loopback functionality), Context enables QoS policy control, Connections expose control of VPI/VCI mappings, Statistics exposes packet and byte counters and Alarms push state change notifications to the controller. Any Tempest switch runs an Ariel server and translates Ariel request to Prospero control requests. Ariel fundamentally was an abstraction layer between the switch silicon and the Prospero control interface.

Caliban Switch Management Interface In addition to network control, Tempest also supported 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 ATM network, and used effective bandwidth measurement techniques to estimate available resources. This admission control scheme provided 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. In addition, 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 SDN

A recent attempt for distributed and scalable network control is the SDN paradigm. SDN Foundations [2012] provides a clean separation between the control and the forwarding plane of a network device. The control plane logic is removed from the net-

work device and implemented in a separate programmable device. A well defined protocol establishes the integration between the two devices.

The SDN approach is inspired by relevant earlier attempts in network programmability(Active Networks and DCAN), but follows an evolutionary approach and provides a mechanism to improve the performance of existing data plane protocols and develop custom forwarding algorithms. The paradigm is motivated by two key observations. Firstly, the evolution of computer networks has collapsed the layers of the OSI and TCP/IP model. Production networks contain middleboxes (e.g. firewall, NAT, layer-5 switches) operating on multiple layers of the network and engaging extensively in layer violation. The SDN paradigm provides a unifying cross-layer control model that matches better to modern network device functionality and integrates under a single interface. Secondly, although network complexity has increased, network devices still use stand-alone configuration mechanisms (e.g. remote logic, CLI interfaces), while network administrators must specialize global network policy into individual network device policies, to fit the existing control abstraction. Furthermore, a series of distributed control protocols remain proprietary, thus reducing device interoperable. SDN enables management centralisation and fast reactive control.

The SDN paradigm was highly successful in the network community and within a few years since its introduction, network vendors started to integrate support in production-level devices. In parallel, the research community has developed a number of SDN architectures which address several network limitations. The OpenFlow protocol is currently the pre-dominant implementation of the SDN paradigm.

OpenFlow protocol

The OpenFlow protocol was originally developed by the OpenFlow Consortium, a collaborative organisation of academic institutes, but soon its development was adopted by the ONF, a standards definition committee comprised of academic institutions and network vendors. OpenFlow has been a highly successful approach to network control. A number of vendors has introduced support of the protocol in their products while there are already OpenFlow deployments in enterprise networks Google [2013]; Kobayashi et al. [2012].

Figure 2.3 presents an example of an elementary OpenFlow topology consisting of

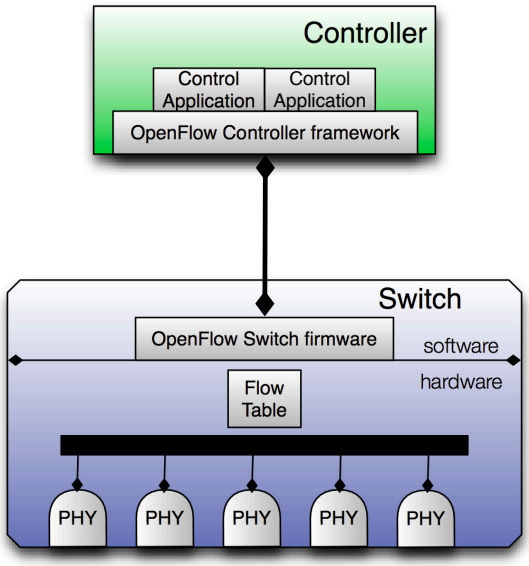


Figure 2.3: A schematic representation of a basic OpenFlow setup

Field	OpenFlow Version
src & dst MAC addr.	1.0 [a]
input port	1.0
VLAN ID	1.0
VLAN PCP	1.0
MPLS label	1.1
MPLS class	1.1
IPv4 src/dst addr.	1.0
IPv4 proto	1.0
ARP opcode	1.0
ARP src/dst IPv4 address	1.0
ARP src/dst MAC address	1.2

Field	OpenFlow Version
IPv4 ToS	1.0
ICMPv4 Type & Code	1.0
TCP/UDP/SCTP src/dst port	1.0
metadata	1.1
IPv6 src/dst addr.	1.2
IPv6 flow label	1.2
ICMPv6 type & code	1.2
ICMPv6 Network Discovery target 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 available packet actions

two entities: the *controller* and the *switch*. The controller implements and disseminates the control logic of the network to one or more switches over a control channel. In addition, the network community has developed a number of programming environments to develop OpenFlow control applications. Such programming environments provide a higher level abstraction over the OpenFlow protocol and reduce the exposure of the developer to the run-time details Berger [2012]; Gude et al. [2008b]; Switch [2013].

this is a simplified description for the protocol model. Different protocol version extend some aspects of the model The switch entity is responsible to transform OpenFlow protocol interactions in data path policy updates. The OpenFlow functionality is split between the data and control layer of the switch. In the data layer, the protocol implements a simple packet processing algorithm. For each packet, the hardware extracts a number of fields and matches them against the entries of one or more *Flow Table*. The Flow Table is a memory abstraction which contains forwarding policy. It comprises of flow entries which conceptually consist of three parts: the flow tuple, the flow statistics and the flow action list. The flow tuple is effectively a flow description using all the significant packet header fields. The flow definition has evolved over the years and Table 2.1 presents the list of supported fields by each protocol version. The protocol associates each flow tuple with a flow mask, permiting field wildcarding. 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. For each packet, if a matching flow is found in the flow table, 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 propagates as an exception to the control layer of the switch.

The control plane of the switch implements the translation of the OpenFlow protocol into data path modifications. It provides access and modification functionality to the switch configuration and the flow table. The switch is able also to propagate autonomously information and exceptions to the controller.

The ensemble of a flow table and a set of switch ports comprises an OpenFlow *datapath* and is the core abstraction for network control. The protocol defines a number of protocol message types to manage datapath resources. OpenFlow control operations can be grouped in two categories: *Forwarding control* and *Switch configuration*. In the rest of the section we present how this functionality is addressed by OpenFlow

messages and how this mechanism is evolved over the different versions of the protocol. We focus our protocol presentation over the 1.0 version of the protocol. ONF has releases three revisions of the protocol, version 1.1, 1.2 and 1.3, which change significantly the protocol specification. Nonetheless, the majority of production systems are predominantly version 1.0 compliant.

Forwarding control OpenFlow provides fundamentally two modes of control: *reactive* and *proactive* control. In the reactive approach, the switch is configured to forward every unmatched packet to the controller, and the controller is responsible to respond with appropriate modification in the flow table. The protocol defines the `pkt_in` message type, that can propagate the header of a packet to the controller, and the `flow_mod` message type, that can manipulate flow table entries. The reactive control approach provides fine control over the traffic dynamics, but, depending on the deployment environment, can introduce significant load on the control plane. In the proactive approach, the controller is responsible to pre-install all required flows in the flow table and avoid any packet handling exceptions. Because this approach lucks data plane feedback, the OpenFlow protocol provides the `stats_req/stats_resp` and `aggr_stats_req/aggr_stats_resp` message types. These message types allow the controller to poll dynamically for flow statistics and infer network resource allocation. In order to ensure atomicity in the protocol, the protocol also define a `barrier_req/barrier_reply` message type, which provides a synch point between the controller and the switch. A `barrier_reply` is send by a switch as a response to a `barrier_req` from the controller, only if all previous operations are processed. Finally, the OpenFlow protocol provides two additional message types to increase users ability to interact with the forwarding plane: 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. The protocol defines flow timers and the switch can remove a flow with an expired timer.

Switch configuration Apart from the control of the flow table, the OpenFlow protocol provides switch configuration capabilities. A controller can use the `switch_config_req/switc` message types, to discover switch OpenFlow operation support. In addition, the proto-

col provides capabilities to control port state 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 the recent revisions of the protocol, the steering committee has introduced the capability to control per port traffic shaping queues.

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

Version 1.1 modifies the main protocol processing pipelne and improves the integration of the protocol with the ASIC functionality. In terms of packet processing, the packet lookup process searches sequentially, instead of 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 define a 64-bit metadata field and the action list is augmented with operations over the metadata field, as well as, over the table search process (e.g. terminate lookup, skip table). *this is shit* Secondly, the protocol introduces a primitive to express multipath support in the protocol. This functionality is exposed through a group table abstraction. Group table entries contain a set of buckets containing flow actions and an assigned output port. A flow action list 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 matching support and manipulation.

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 defines a new TLV format for flow Match definition, in order to relax the OpenFlow tuple definition. Finally, the protocol defines a model for efficient multi-controller switch functionality, in an effort to improve control resilience.

Version 1.3 introduces protocol support of QoS control. The protocol defines a new

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

Table 2.3: List of OpenFlow controller 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

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 parallelize `pkt_in` transmission, and defines a mechanism to support message fragmentation.

Because the recent versions of the protocol has become complex, the vision of future OpenFlow deployment, is to differentiate switch abstraction and develop diverse product ranges that will provide optimized support for a subset of the protocol.

2.4 SDN applications

The introduction of the SDN paradigm proved to be extremely successful not only in the research community, but also in the network industry. Already a number of vendors provide production-level hardware support for SDN (Table 2.4, and the SDN community provides OpenFlow support for most programming languages 2.3. In order to exhibit the class of problems which can be tackled by appropriate design of the

control plane, in this section we present applications of the SDN paradigm in different network problems.

2.4.1 Network Virtualisation

The introduction of OS virtualization and the subsequent rise of the cloud computing paradigm has created new opportunities for computer science to reduce infrastructure costs and create new highly resilient system architectures. Although platforms like Xen provide strong guarantees on resource allocation and performance isolation on the OS level, there is still a significant gap in network virtualisation. The introduction of the SDN paradigm provides novel capabilities to the research community to develop new network abstractions and expose direct network control to tenants, while assuring the correct functionality of the network.

The first attempt to enable network virtualisation in the data center was through the FlowVisor project Sherwood et al. [2010a]. FlowVisor function as a OpenFlow proxy and sits between the OpenFlow switches and the OpenFlow controller. By appropriate message processing, a network topology discovery mechanism and a resource management policy, FlowVisor aggregate control over a set of switches and exposes a single virtual switch abstraction. The administrator can partition the OpenFlow tuple and delegate network control to different controllers, thus allowing data center tenants to control traffic for their own subnet and VM group. FlowVisor is currently adopted by BigSwitch and developed as a product. In Corin et al. [2012] the authors develop an extension for the FlowVisor platform, which enables topology virtualisation capabilities. In Drutskoy et al. [2013] the authors present FLowN, a network virtualisation mechanism which provides full control of OpenFlow tupple to all tenants and multiplexes user traffic using VLAN tags. FlowN, in addition, removes the latency incurred by control proxy by encapsulating the control transformation process in the programming library. Furthermore, state synchronisation is implemented through an SQL database, which is used by the programming library to transform OpenFlow messages. Finally, Nicira has develop Nicira Network Virtualization Platform (NVP), which claims to provide full network virtualisation and control isolation. Nonethless, the product is proprietary and implementation details remain undisclosed.

2.4.2 Security and Access Control

As we have discussed in the introduction of this thesis, network security has become a significant requirement for modern network operability. In order to improve network security, we require novel control plane architectures which are inherently secure. SDN provides a mechanism to enable fast prototyping of secure network architectures, while the high programmability of the controller allows traffic-inspecting security application integration with the control logic of the network. In an early attempt to define the SDN abstraction, researchers in Stanford developed the Ethane system Casado et al. [2007]. Ethane defines a simple policy expression language which uses as first-class object user identities and network services, and defines the interconnection ability between the two. SDN provides also the means to develop lossless, highly performant and programmable network trace collection mechanisms. In Ballard et al. [2010], the authors present OpenSAFE, a policy expression language which allows a network manager to define policies to intercept and inspect network traffic, enabling a highly flexible and dynamic network monitoring applications. Finally, in Porras et al. [2012], the authors present FortNOX, a mechanism which enables secure interaction between the control applications of a network. FortNOX enables users to assign priorities and roles to controlling applications and implements a simple resolution mechanism when two applications introduce contradicting network policies.

2.4.3 Load Balancing

The introduction of network services with global audience has created a requirement for IP indirection mechanisms, to distribute network load between servers in server farms and cloud computing infrastructures. A common approach to address this problem is through proprietary load balancing devices that function on the gateway of the server farm and distribute dynamically user request between servers. The introduction of the SDN abstraction integrates this indirection functionality in the network fabric. In Wang et al. [2011], the authors present a framework for in-network load balancing functionality. The system preinstall wildcard rules to forward new flows to a specific server. As the server load changes, the control dynamically modifies the wildcard coverage and redirects new flows to less utilised servers. In Handigol et al. [2009b, 2010], the author present a reactive approach to the problem. In the Plug-and-Serve architec-

ture, the controller is responsible to handle each new connection and forward traffic to a specific server, based on the allocation algorithm and the current load of the system.

2.4.4 Inter-domain and Intra-domain Routing

In the context of network control, the SDN paradigm has also been proposed in order to improve the performance of existing control plane protocols. In Rothenberg et al. [2012], the authors present an integration of the Quagga Quagga routing framework with the OpenFlow protocol, in an effort to revisit the Route Reflector idea in BGP routing Bates et al. [2006]. Furthermore, In an effort to reduce the impact of routing instabilities of the BGP protocol, the authors in Kotronis et al. [2012] propose a mechanism to offload BGP routing to third party entities and use the OpenFlow abstraction to propagate resulting FIBs to the outsourced-network devices.

The programmable nature of the SDN paradigm and its ability to centralise network control has also motivated explorations on network state compression. In Sarrar et al. [2012], the authors present a library which exposes a FIB abstraction over the OpenFlow abstraction to routing applications. The library at run-time analyses the resulting FIB, as well as, the traffic pattern and calculates an minimum set of flow entries which can server a specific subset of the traffici on the fast-path of the network. In a similar effort, authors in Yu et al. [2010a] present DIFANE, a routing mechanism which uses valiant inter-routing, in an effort to compress the average FIB table size within the network. The proposed mechanism uses the OpenFlow abstraction in order to partition effectively the network and disseminate forwarding state across the network devices.

2.4.5 Network Management

One of the core goals of the SDN paradigm is to provide the development environment for innovative management mechanisms which ease the network control task. One of the first application of the SDN paradigm was introduced by Koponen et al. [2010] within the context of the Onix control platform. Onix provides a centralised control abstraction to network developers, over which they can program their network control logic. The abstraction encapsulates a control state distribution mechanism which enables seamless distributed execution of code.

In a similar effort a number of novel programming languages has be proposed to formalise the control plane requirements. In Foster et al. [2011] the authors present the Frenetic programming language, a declarative language to define network control policy. The language encapsulates in the programming model significant network control programming requirements, like race conditions abstraction mechanisms. Furthermore, in Monsanto et al. [2012] the authors describe NetCore, a novel policy expression language which enables users to express their control logic. At run-time the policy adapts to the network traffic and topology in an effort to optimizes the installed state within the network. The NetCore abstraction is further enhanced with the ability to provide verifiable control policies in the network, as presented in Guha et al. [2013]. In addition, in Monsanto et al. [2013] the authors define a framework build on top of the NetCore language which enables seamless integration of the control logic between different control applications and reduces the complexity to integrate individual control applications in the network. Such control abstraction have also been discuss in the context of network control logic updates. In Reitblatt et al. [2012], the authors present a verifiable mechanism to upgrade the network control logic without any network disruption. Finally, in Voellmy et al. [2012], the authors present a novel declarative policy language which builds on top of the reactive functional programming model and provides a highly flexible architecture to develop novel control management systems.

2.4.6 Energy

In the recent years energy consumption has become an important measure of efficiency for a system architecture. Currently the network is estimated to contribute approximately 20% of the total power consumption of a data-center. Unlike CPUs which can reduce power consumption when idle, network cards exhibit a constant high power consumption, due to the constant requirement for medium sensing and physical layer frame synchronisation. As a result the most efficient mechanism to reduce network power consumption is to shutdown interfaces. In Heller et al. [2010], the authors present a power-aware control plane architecture, build 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

The SDN paradigm, as well as, the OpenFlow abstraction provide novel mechanisms to troubleshoot a network. In Wundsam et al. [2011], the authors present a mechanism to intercept and record traffic flows from a network. Recorded flows can be replayed within the network, in order to detect policy misconfigurations which create functionality problems. In a similar effort, in Handigol et al. [2012b] the authors present ndb, a control plane debugger which enables users to register data plane breakpoint conditions and receive rich control plane run-time informationi when this conditions are met. Finally, a number of applications have been proposed to enable control plane monitoring in order to detect potential networks misconfigurations. In Khurshid et al. [2012], the author present an OpenFlow proxy service which detect flow modifications which create forwarding policy inconsistencies. In Canini et al. [2012], the authors present a model checking mechanism which uses symbolic execution to detect flow modifications which create significant inconsistencies in the data plane forwarding. Debugging applications of the OpenFlow abstraction have been also proposed in the context of home networks. In Calvert et al. [2010], the authors proposed a network controller which enables precise network logging by the home router, in order to aid ISPs to troubleshoot connectivity problems.

2.4.8 Resource Control

The high reactivity of the SDN paradigm enables fine level resource control integrated with the control plane. Already a number of control designs applicable in the datacenter environment have been proposed by the research community. In Al-Fares et al. [2010] the authors present Hedera, a novel control architecture for datacenters which improves the network utilisation in comparison to the popoular ECMP Hopps [2000] network load balancing mechanism. The system uses the flow statistics mechanism provided by the OpenFlow abstraction and at run time discovers progressively network-limited flows which are rerouted over underutilised network paths. In a similar effort, the authors in Benson et al. [2011] present a network load distribution mechanism with a higher granularity on short-lived flows. Furthermore, in Even et al. [2012] the authors propose a network embedding mechanism which is able to define job assignments in a datacenter and provide strong guarantees on network resources.

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

2.4.9 Mobility

The SDN paradigm has found excellent applicability in addressing the increased control plane requirement in mobile networks. In Huang et al. [2010] the authors present PhoneNet, an application development framework which provides the ability to applications to setup multicast groups in the network through the controller. In Yap et al. [2009, 2010] the authors present OpenRoads, a network control plane for wireless network which enables seamless Access Point (AP) hand-overs for devices, as well as, network virtualisation. Finally, the authors in Li et al. [2012] discuss SDN abstraction requirements for cellular networks.

2.4.10 Network Simulation

The capability of the SDN paradigm for fast network functionality prototyping provides a novel experimentation mechanism for network researchers. Currently a number of large-scale shared infrastructures are built and maintained by research organisation. The GENI testbed Global Environment for Network Innovations (GENI) [2013] in the US funded by the NSF and the Ofelia testbed Ofelia [2013] in the EU funded by the FP7 framework provide network and computing resource to run large-scale experiments for free. Furthermore, a number of OpenFlow-based frameworks has been developed for scalable network experimentation. In Erickson et al. [2011] the authors present Virtue, a large-scale emulation framework which uses the Xen virtualization platform and the OpenFlow protocol to enable user experimentation with network topologies and VM placements in a multi-tenant data center environment. In addition, the Mininet platform Handigol et al. [2012a]; Lantz et al. [2010] has been developed by researchers in Stanford to enable scalable experimentation with the OpenFlow protocol. Mininet uses the LXC LXC [2013] linux user-space virtualisation framework to

run multiple virtual machine in a single host and experiment with network functionality. Mininet has been used in the Computer Science department in Stanford to aid the introduction of students with recent research efforts in computer networks McKeown et al. [2012].

2.5 Conclusions

In this Section we have provided an in-depth analysis of network control mechanisms. We motivate our discussion in network control by presenting a generic architectural model for network devices and focus on the inherent limitations for network control. Furthermore, we present the current control plane mechanisms in production networks and, motivated by the limitations of these mechanisms, we present three control mechanisms, proposed by the research community, which aim to improve the respective limitations. Finally, we provide 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

In this thesis we develop mechanisms to scale computer network functionality, by redesigning the control plane. In our exploration we employ extensively he SDN design paradigm and the OpenFlow protocol abstraction. We chose the SDN design approach due to two core functional properties. On one hand, the clean separation of the control and the forwarding plane in a network is inherently backwards compatibility with existing network applications and interoperable with existing network devices. The evolution of the control plane of a network doesn't affect forwarding plane protocol support, while SDN supports progressive deployment in production networks without any forwarding performance penalty. Secondly, the OpenFlow control abstraction is sufficiently generic to support a diverse set of control approaches. The protocol supports reactive and proactive control schemes, while the flow definition granularity is dynamically controlled to match the application requirements. In the following chapters of this thesis we employ the OpenFlow protocol to address two diverse network scalability problems.

In this chapter we present an extensive performance analysis of the control scalability of available SDN technologies. Our exploration aims to provide an in-depth presentation of the limitations of existing implementation efforts and understand their impact in forwarding plane performance, as well as, provide a set of tools which enable SDN developers to study the performance of their network designs. The work focuses on implementations of version 1.0 of the OpenFlow protocol, the only production-level protocol instantiation of the SDN paradigm. We conduct our analysis using 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 to characterise the performance of elementary OpenFlow 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, developers are able to import OFLOPS switch profiles in 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 the OFLOPS platform (Section 3.2). We select a number of off-the-self OpenFlow switches (Section 3.3) and run against them a number of measurement experiments, in order to assess the elementary protocol interaction performance (Section 3.4). Furthermore, we present the SDNSIM platform (Section 3.5) and its design approach (Section 3.7). Finally, we assess the performance of the SDNSIM implementation along with a measurement study of control scalability over the fat-tree topology (Section 3.8) and conclude (Section 3.10).

3.1 Network Control Micro-Benchmark

Despite the recent introduction of the SDN paradigm, the research community has already proposed a wide range of novel control architectures Handigol et al. [2009a]; Sherwood et al. [2010b]; Yu et al. [2010b]. These architectures address significant problem of modern networking, but their deployment in production environments is not straightforward. Computer network have become a vital asset for modern enterprises, and high availability and performance are critical. Modifying established network control mechanisms must ensure these requirements and requires extensive testing and performance characterisation. An example of an early OpenFlow-based SDN deployment that faced significant problems in a production environment is reported in Weissmann and Seetharaman [2011]. The authors describe their experience in deploying the first OpenFlow production network in the Computer Science department in Stanford University. In their article, they point out that the initial deployment exhibited significant performance and reliability problems. The deployed hardware switching platform couldn't support OpenFlow message processing at high rates. For example, in the morning a significant number of users would plug simultaneously their

computers to the network, creating a large burst of new network flows. Because the reactive control approach they employed requires the generation of a `pkt_in` message for each new connection, the switch control CPU processing capacity couldn't cope with the high utilisation and either a number of flows would be dropped or the control channel would become unresponsive. In the SDN application development toolchain, there is a lack of an established and global mechanism to assess OpenFlow mplementation scalability. For traditional switch devices, the network community employs as performance metrics, the switch fabric non-blocking processing capacity and latency and MAC table size. In contrast, the control decoupling of the SDN paradigm expands the ways that the controller interacts with a switch device, while these interactions become a significant requirement for the functionality of the switch. As a result, the performance characterisation of SDN forwarding devices becomes non-trivial.

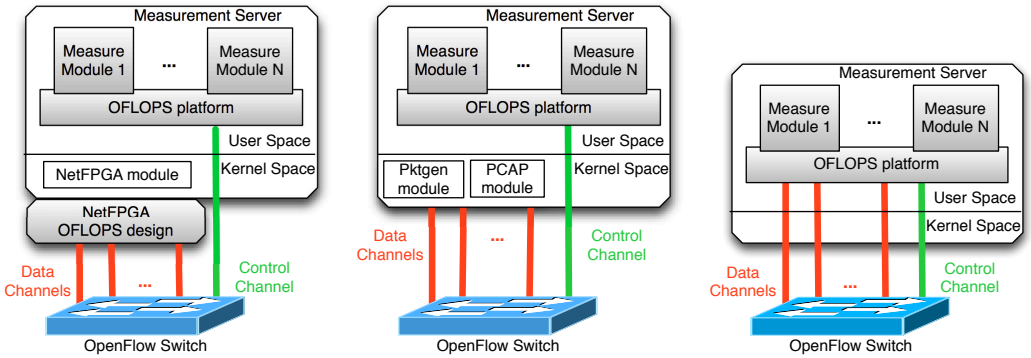
In order to fulfil the requirement for OpenFlow switch performance evaluation, we developed OFLOPS [1], a measurement framework enabling rapid development of performance tests for hardware and software OpenFlow switch implementations. To better understand the behaviour of the tested OpenFlow implementations, OFLOPS combines measurements from the OpenFlow control channel with data-plane measurements. To ensure sub-millisecond-level accuracy of the measurements, we bundle the OFLOPS software with specialized hardware in the form of the NetFPGA platform[2]. 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 characterization accuracy, noise suppression and precision. Performance characterization is not 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. Further, current controller frameworks, like Gude et al.

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

[2] `http://www.netfpga.org`



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

Figure 3.1: OFLOPS measurement setup. The framework provides integration with: 1) a high accuracy NetFPGA packet generator hardware design (Figure 3.1(a)), 2) kernel space `pktgen` traffic generation module and PCAP packet capturing module 3.1(b), 3) User space packet capture and generation threads (Figure 3.1(c)

[2008a]; Switch [2013], target production networks and incur 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 programing interfaces, that require multiple data structure allocation and modification during packet processing. The processing time of a specific OpenFlow message, especially at high rates, is subject to multiple aspects, like the thread scheduling policy and the asynchronous library execution logic Jarschel et al. [2012]. Measurement noise suppression in the control plane requires 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.

The OFLOPS design philosophy aims to develop a low overhead abstraction layer that allows interaction with an OpenFlow-enabled device over multiple data channels. The platform provides a unified system that allows developers to control and receive information from multiple control sources: data and control channels as well as SNMP to provide specific switch-state information. For the development of measurement experiments over OFLOPS, the platform provides an event-driven, API that allows de-

velopers to handle events programatically in order to implement and measure custom controller functionality. The current version is written predominantly in C. Experiments are compiled as shared libraries and loaded at run-time using a simple configuration language, through which experimental parameters can be defined. A schematic of the platform is presented in Figure 3.2. Details of the OFLOPS programming model can be found in the API manual OFLOPS [2011].

The platform is implemented as a multi-threaded application, to take 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 consists of the following five threads, each one serving specific type of events:

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 provides the ability to control concurrently multiple data channels to the switch. Using a tight coupling of the data and control channels, programers can understand the impact of the measurement scenario on the forwarding plane. To enable our platform to run on multiple heterogeneous platforms, we have integrated support for multiple packet generation and capturing mechanisms. For the packet generation functionality, OFLOPS supports three mechanisms: user-space (Figure 3.1(c)), kernel-space through the pktgen kernel module module Olsson [2005b] (Figure 3.1(b)), and hardware-accelerated through an extension of the design of the NetFPGA Stanford Packet Generator Covington et al. [2009] (Figure 3.1(a)). For the packet capturing and timestamping, the platform supports both the pcap library and the modified NetFPGA design. Each approach provides different precisions and different impacts upon the measurement platform.

In order to assess the accuracy of the traffic capturing mechanism, we use the topology depicted in Figure 3.2 to measure precision loss in comparison to a DAG card Endace [2012]. To calculate precision, we use a constant rate 100 Mbps probe of small packets for a two minute period. The probe is duplicated, using an optical wiretap with negligible delay, and sent simultaneously to OFLOPS and to a DAG card. In

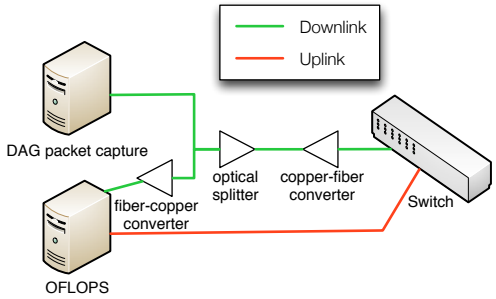


Figure 3.2: Measurement topology to evaluate capturing mechanism precision.

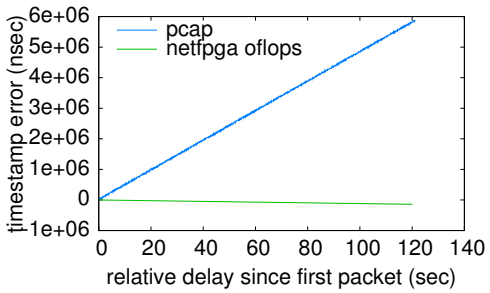


Figure 3.3: Evaluating timestamping precision using a DAG card. PCAP timestamps exhibit a significant drift in the order of microseconds, in comparison to the NetFPGA hardware design

Figure 3.3, we plot the differences of the relative timestamp between each OFLOPS timestamping mechanism and the DAG card for each packet. From the figure, we see that the PCAP timestamps drift by 6 milliseconds after 2 minutes. On the other hand, the NetFPGA timestamping mechanism has a smaller drift at the level of a few microseconds during the same period.

3.3 Measurement Setup

At the end of 2009, the OpenFlow protocol specification was released in its first stable version 1.0 OpenFlow Consortium [2009b], the first recommended version implemented by vendors for production systems. The switch performance analysis was contacted 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 top-of-rack-style switching hardware, during that time. Details regarding the CPU and the size of the flow table of the switches are provided in Table 3.1. 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 OpenFlow protocol reference 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 Dum0, a configuration which is inherited by major Cloud providers, like Amazon and Rackspace. Secondly, several hardware switch vendors use Open vSwitch as the basis for the development of their own OpenFlow-enabled firmware. Open vSwitch development team has standardised a clean abstraction for the control of the 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 study Open vSwitch alongside 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.

Switch	**CPU**	**Flow table size**
Switch1	PowerPC 500MHz	3072 mixed flows
Switch2	PowerPC 666MHz	1500 mixed flows
Switch3	PowerPC 828MHz	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.

In order to conduct our measurements, we setup OFLOPS on a dual-core 2.4GHz Xeon server equipped with a NetFPGA card. For all the experiments we utilize the NetFPGA-based packet generating and capturing mechanism. 1Gbps control and data channels are connected directly to the tested switches. We measure the processing delay incurred by the NetFPGA-based hardware design to be a near-constant 900 nsec.

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 switch OpenFlow implementation, we develop 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 OpenFlow flow table along with its impact on the data plane, (Section 3.4.3) (4) the monitoring capabilities provided by OpenFlow (Section 3.4.4), and (5) the impact of interactions between different OpenFlow operations (Section 3.4.5).

3.4.1 Packet Modifications

The OpenFlow 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 generate 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 causing packet queuing. The flow table is initialized with a flow that applies a specific action on all probe packets and the

Mod. type	Switch 1			ovs			Switch 2		
	med	sd	loss%	med	sd	loss%	med	sd	loss%
Forward	4	0	0	35	13	0	6	0	0
MAC addr.	4	0	0	35	13	0	302	727	88
IP addr.	3	0	0	36	13	0	302	615	88
IP ToS	3	0	0	36	16	0	6	0	0
L4 port	3	0	0	35	15	0	302	611	88
VLAN pcp	3	0	0	36	20	0	6	0	0
VLAN id	4	0	0	35	17	0	301	610	88
VLAN rem.	4	0	0	35	15	0	335	626	88

Mod. type	Switch 3			NetFPGA		
	med	sd	loss%	med	sd	loss%
Forward	5	0	0	3	0	0
MAC addr.	-	-	100	3	0	0
IP addr.	-	-	100	3	0	0
IP ToS	-	-	100	3	0	0
L4 port	-	-	100	3	0	0
VLAN pcp	5	0	0	3	0	0
VLAN id	5	0	0	3	0	0
VLAN rem.	5	0	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).

processing delay is calculated as the difference between the transmission and receipt timestamps provided by the NetFPGA. 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.

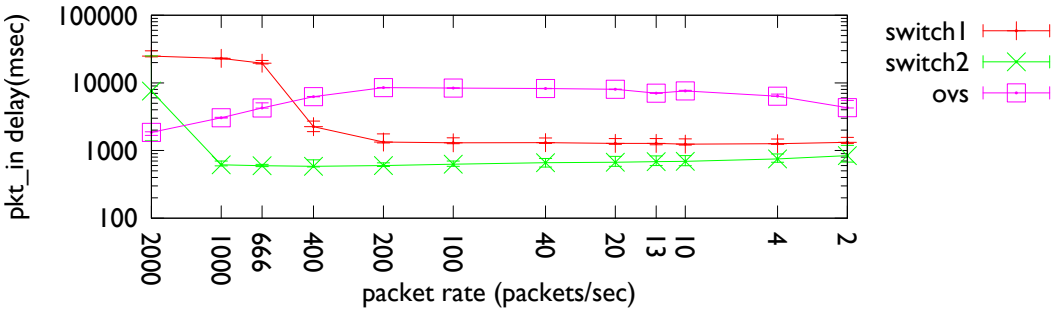
We observe significant differences in the performance of the hardware switches due in part to the way their firmware implements packet modifications. Switch1, with its production-grade implementation, handles all modifications in hardware; this explains its low packet processing delay between 3 and 4 microseconds. 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 two orders of magnitude

higher packet processing delay and variance. 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 when the flow actions are supported in hardware.

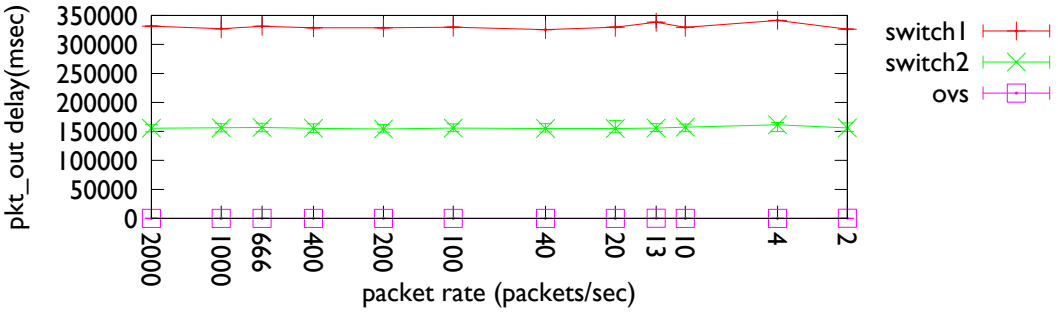
We conducted a further series of experiments with variable numbers of packet modifications in the flow action list. We observed, that the combined processing time of a set of packet modifications is equal to the highest processing time across all individual actions in the set (e.g. Switch2 requires approximately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

3.4.2 Traffic Interception and Injection

OpenFlow protocol permits a controller to intercept or inject traffic from the control plane. Packet interception is fundamental for reactive control applications, while packet injection enables control application interaction with network hosts. Nonetheless, the interception mechanism in OpenFlow has been characterised as a significant bottleneck for the control plane in early OpenFlow protocol deployments Kobayashi et al. [2012]. This is a direct consequence of the silicon design in current OpenFlow switches, that develop such functionality over a low-bandwidth exception-notification channel (Section 2.1). In order to characterise these functionalities, we design two simple experiments. For packet interception, we remove all entries from the switch flow table and send a measurement probe of small packets (100 bytes) on one of the data channels. We measure the delay between the time the packet was transmitted on the data channel and the time the controller received the equivalent `packet_in` message. For packet injection, we transmit `packet_out` messages over the control channel and measure the delay to receive the packet on a data channel. In Figure 3.4, we plot the median packet processing latency for `packet_in` and `packet_out` messages. We omit in this experiment Switch 3 as these functionalities incur high CPU utilisation and, after a few seconds of traffic, the control channel becomes unresponsive. For



(a) `packet_in` message latency



(b) `packet_out` message latency

Figure 3.4: Latency to intercept and inject a packet using the OpenFlow protocol

`packet_out` messages, all implementations rate limit their transmission through the TCP advertised window of the control connection and as a result the latency is near constant. We observe that hardware switch exhibit high latency (150 msec for Switch2 and 350 msec for Switch1), in comparison to Open vSwitch (approximately 0.1 msec). For `packet_in` messages, we observe diverse behaviours between hardware switches at high packet rates. For Switch1, packet loss and latency becomes note-worthy beyond 400 packets/sec, while the switch can process up to 500 packets/sec. For Switch2 latency and packet loss are significantly lower and stable. Switch2 incurs high processing latency beyond 2000 packets/sec. Open vSwitch, has a high but stable latency for any tested data rates.

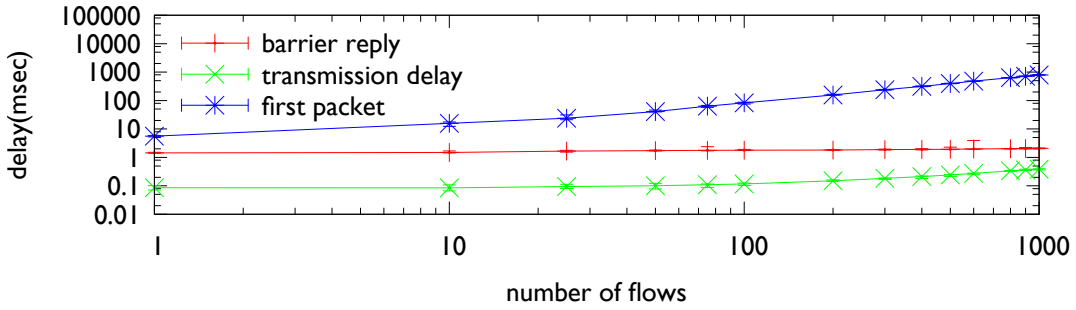
3.4.3 Flow Table Update Rate

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 provide 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].

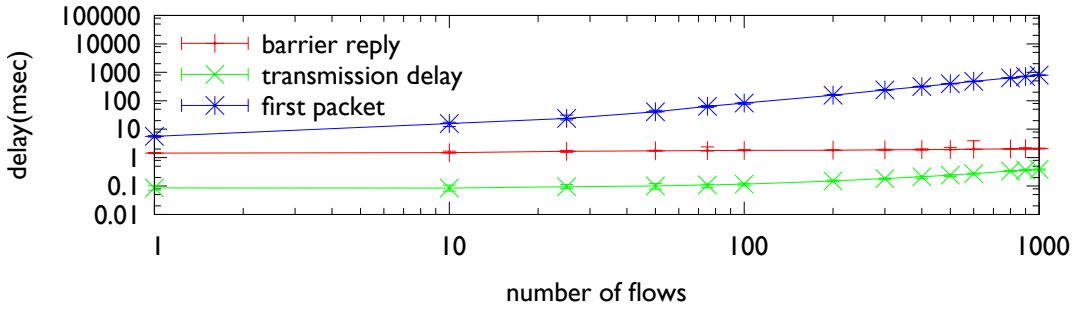
OpenFlow provides a mechanism to define barriers between sets of commands: the `barrier` command. According to the OpenFlow specification OpenFlow Consortium [2009b], the `barrier` command is a way to be notified that a set of OpenFlow operations has been completed. Further, the switch has to complete 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 check the behavior of the tested OpenFlow implementations, finding variation among them. For Open vSwitch and Switch1, Figure 3.5 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 relies on a stream of packets of 100 bytes at a constant rate of 100 Kpackets/sec (10Mbps) that targets the newly installed flows in a round-robin manner. The probe achieves sufficiently low inter-packet periods in order to accurately measure the flow insertion time.

In Figure 3.5, we show three different times. The first, *barrier notification*, is derived by measuring the time between when the **first insertion command** is sent by the OFLOPS controller and the time the `barrier` notification is received by the PC. The second, *transmission delay*, is the time between the first and last flow insertion commands are sent out from the PC running OFLOPS. The third, *first packet*, is the time between the **first insertion command** is issued and a packet has been observed for the last of the (newly) inserted rules. For each configuration, we run the experiment 100 times and Figure 3.5 shows the median result as well as the 10^{th} and 90^{th} percentiles,



(a) Open vSwitch (log-log scale)



(b) Switch1 (log-log scale)

Figure 3.5: Flow entry insertion delay: as reported using the `barrier` notification and as observed at the data plane.

although the variations are small and cannot be easily viewed.

From Figure 3.5, we observe that even though the *transmission delay* for sending flow insertion commands increases with their number, this time is negligible when compared with data plane measurements (*first packet*). Notably, the *barrier notification* 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 notification* is implemented does not reflect the time when the hardware flow-table has been updated.

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

Flow insertion types

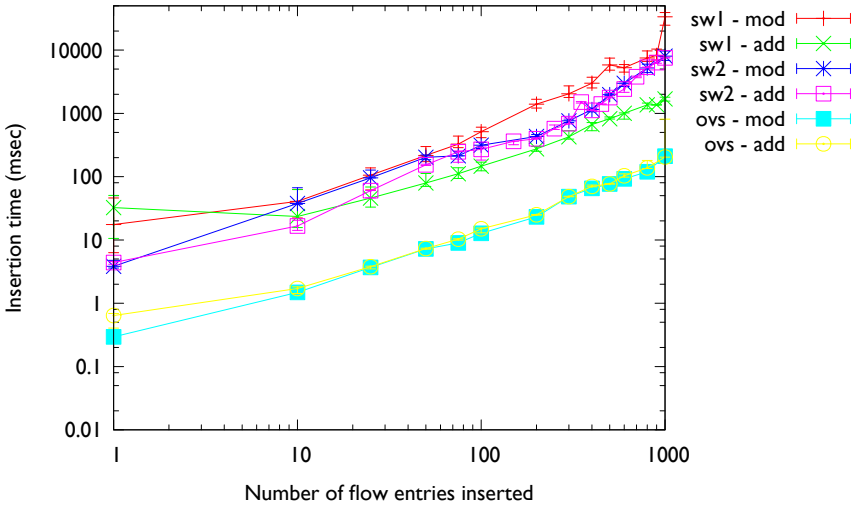


Figure 3.6: Delay of flow insertion and flow modification, as observed from the data plane (log-log scale).

We now distinguish between flow insertions and the modification of existing flows. With OpenFlow, a flow rule may perform exact packet matches or use wild-cards to match a range of values. Figure 3.6 compares 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 requires re-ordering of existing entries. In Gupta and McKeown [2001], the authors characterise the update complexity of a TCAM to be 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, wild-carded entries, requiring support for variable length lookup, must be handled by specialized memory modules, such as a TCAM. With such possible choices and range of different experiments, the flow insertion times reported in Figure 3.6 are not generalizable, but rather depend on the type of insertion entry and implementation.

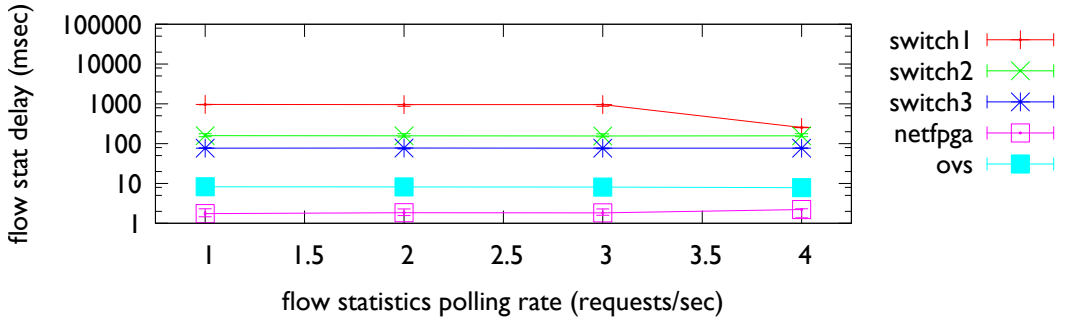
3.4.4 Flow Monitoring

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.

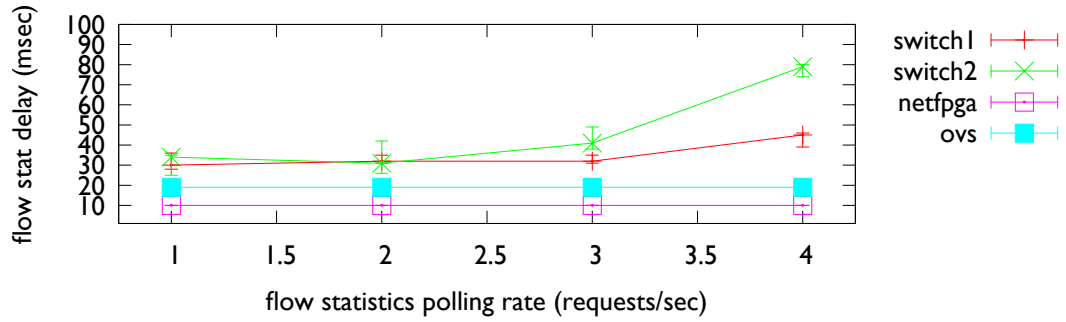
We now test the performance implications of the traffic statistics reporting mechanism of OpenFlow. Using OFLOPS, we install flow entries that match packets sent on the data path. Simultaneously, we start sending flow statistics requests to the switch. Throughout the experiment we record the delay getting 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 3.7(a) reports the time 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 3.7(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.

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 behavior of the exclusively hardware switches whose CPU time increases proportionally with the rate of requests. We observe two types of behavior from the hardware switches: the switch has a high CPU utilization, answering flow-stats requests as fast as possible (Switch2), or the switch



(a) Reply time.



(b) CPU utilization.

Figure 3.7: Time to receive a flow statistic (median) and corresponding CPU utilization.

delays responses, avoiding over-loading its CPU (Switch1). Furthermore, for Switch1, we notice 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 the OpenFlow protocol is its ability to provide network control applications with, e.g., flow arrival notifications from the network, while simultane-

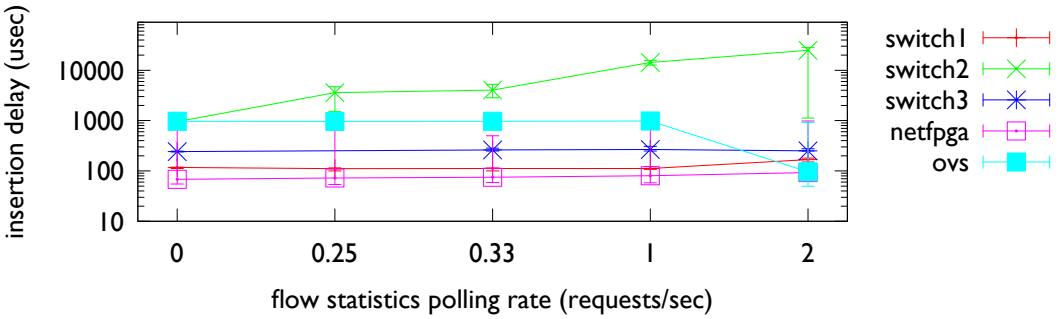


Figure 3.8: Delay when updating flow table while the controller polls for statistics.

ously 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. [2009a]; Yap et al. [2009]. Using OFLOPS advanced measurement instrumentation, we develop a test scenario of dynamic network control, in order to understand the behavior of the switch control and data plane. In this scenario, we emulate the simultaneous querying of traffic statistics and modification of the flow table. More specifically, we extend Section 3.4.3 by showing how the mechanisms of traffic statistics extraction and table manipulation interact. Specifically, we initialize the flow table with 1024 exact match flows and measure 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 10Mbps packet probe to monitor the data path, and polls every 10 seconds for SNMP CPU values.

In this experiment, we control the probing rate for the flow statistics extraction mechanism, and we plot the time necessary before the modified flows become active in the flow table. For each probing rate, we repeat the experiment 50 times, plotting the median, 10^{th} and 90^{th} percentile. In Figure 3.8 we can see that, for lower polling rates, implementations have a near-constant insertion delay comparable to the results of Section 3.4.3. For higher probing rates on the other hand, Switch1 and Switch3 do not differ much in their behavior. 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 3.7(b)). Finally, Open vSwitch exhibits a marginal decrease in the median insertion delay and at the same time an increase in its variance. We believe this behavior 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 OpenFlow Macro-experimentation

OFLOPS, along with Cbench Cbench [2010], provides a sufficient toolchain to profile functionality of OpenFlow building blocks. Nonetheless, in order to understand the impact of scalable control in a computer network, we require mechanisms to transform OpenFlow building block performance profiles into network-wide performance measurements, under specific traffic patterns and network topology. A common practice to reason about performance and correctness of a network architecture relies on the design of experiments that reproduce specific network-wide functionalities. In the related literature on network experimentation there have been two main implementation approaches : *realistic-testbed* and *simulation*.

Realistic testbeds reproduce in full detail the properties of the deployment environment. They provide an optimal measurement environment with complete control over the parameter of the experiment and optimal time scalability. Nonetheless, realistic testbeds incur significant resource and configuration overhead, which scale sub-optimally with respect to the experiment size. For example, setting up a realistic testbed for datacenter network experimentation requires a significant number of machines and network devices, careful interconnection planning and targeted metrication and analysis of the resulting system. In an effort to improve resource scalability for realistic testbeds, the research community provides a number of shared testbeds. Shared testbeds employ techniques such as virtualization and statistical multiplexing to scale resource utilization in a multi-tenant experimentation platform Emulab [2000]; PlanetLab [2007]. However, shared testbeds are not always a good fit for network experiments. In such testing environments, there is limited resource control, while resource sharing may introduce measurement noise, which is not always detectable and removable.

In the simulation approach, researchers replace parts of the functionality of the system with simplified models Issariyakul [2012]; Varga and Hornig [2008]. Simulation reduces the complexity of large scale network experiments, and provides resource scalability. Nonetheless, the scaling property has inherent limitations. Firstly, the fidelity

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.

of the results depends greatly on the validity of the model assumptions. Secondly, in order to simulate network experiments, users usually need to readjust the logic of their experiments to fit with the abstraction of the underlying models. 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.

SDNSIM[1] is a novel network experimentation framework, that bridges the two aforthmentioned approaches. The framework is written in OCaml, a high performance functional language, and extends the functionality of the Mirage [2] library OS. Developers can describe network functionality over the Mirage OS abstraction, and at compilation transform the experiment definition in 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*, translates the experiment definition in Xen-based emulation of the experiment.

3.6 Mirage Library OS

In SDNSIM we use the Mirage framework to provide an application development environment. Mirage is a a cloud-service development framework written predominantly in OCaml, where applications are single purpose appliances that are compile-time specialised into standalone kernels deployable on the Xen platform. We chose to use rhe Mirage framework due to two fundamental functional properties: low memory footprint and simplified integratability. Mirage revisits the idea of library OS; OS func-

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

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

tionality is separated into logical modules and added to an appliance only if the code expresses an explicit dependency. As a result, a Mirage application will integrate in a VM only the required OS functionality, resulting in compact OS images with minimized memory requirements. In Table 3.3, we present the compiled image size for a set of Mirage applications.

Furthermore, Mirage OS provides a simple development API for systems programming. This functionality is implemented by two core modules, named *Net* and *OS*. The OS module provides basic thread scheduling, device management and device IO functionality, while the Net module provide a network stack supporting ICMP, IP, TCP, UDP and DHCP protocols through a socket-like programming API. The interface of the core Mirage API is sufficiently generic to enable development of complex distributed services, while sufficiently simple to integrate Mirage functionality over a wide range of platforms. Specifically, a Mirage application can compile into a Xen image, a linux application and a nodeJS executable. Currently, the platform is also ported to the FreeBSD kernel space and the BareMetal OS Return Infinity [2013], a high performance assembly-code OS.

3.7 SDNSIM design

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<topology module="Simple_tcp_test" backend="ns3-direct"
   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 developer perspective, SDNSIM consists of a single executable that functions as an OCaml build system. Developers implement host functionality as Mirage applications, and use a single XML file to describe the network topology and assign functionality to network nodes. A sample XML file is presented in Listing 3.1. The configuration describes a simple client (host1) - server (host2) configuration. A minimum experiment definition must define the core code module (topology@module), the target executable (topology@backend) and the duration of the experiment (topology@duration). In order to define a host, SDNSIM uses a host XML entity. Each host entity must define a host name (node@name) and a host main function (node@main), while a number of named parameters (node/param) can be passed to the main function. The link XML entity defines a link between two hosts (link@src, link@dst) along with the device (link@queue_size, link@pcap) and propagation properties (link@rate, link@delay), using the link XML entity. Finally, links can also be used to integrate external network interfaces into the simulation, in order to allow the experiment to interact with applications outside of the experiment scope. We enable this functionality using the link_ext XML entity which is similar to the link entity, but instead of src and destination attributes, it uses the host and dev entities to define the host to which the entity will be attached. In addition, an ip attribute is used to define the address of the device.

The functionality of a node in the SDNSIM platform can be split in 3 layers. A simple representation of the architecture of an SDNSIM host is depicted in Figure 3.9. The top layer contains the application logic of the host. This layer is defined by the developer and encodes the traffic model and the network forwarding logic of the experiment. In order to allow realistic traffic patterns, SDNSIM reuses all the application

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

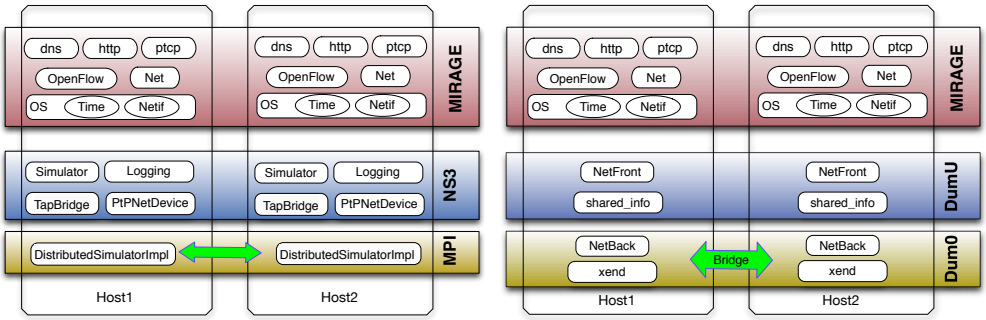
Table 3.4: System facilities provided as Mirage libraries.

libraries supported by the Mirage platform, reported in Table 3.4. In addition, we modify the OpenFlow processing code and introduce a latency control mechanism. Using the rate control mechanism, SDNSIM can simulate and emulate switch and controller models, measured through the OFLOPS and Cbench tools. We also have implemented in OCaml the pttcp tcp test tool Pratt [2002] for model driven traffic generation.

The middle layer of the host architecture, contains the network and the OS libraries of the Mirage platform. This layer provides OS functionality to the application layer. Finally, the lower layer of the host architecture, provides integration of the Mirage OS with the execution environment of the experiment. Currently, SDNSIM supports two execution environments: the *ns-3* simulation platform and the *Xen* virtualisation platform. These two execution environments are highly heterogeneous and employ different execution models. In the rest of this section we present details on the integration of SDNSIM experiments with each execution environment. Since SDNSIM aims for high network accuracy, we focus our presentation on how integration occur on network functionality and time.

3.7.1 Xen backend

An SDNSIM emulation host uses a simple PV Boot mechanism. At startup SDNSIM initializes a VM with a single virtual CPU, loads the Xen event channel and jumps to the main function of the OS module which runs the thread scheduling loop. SDNSIM uses the OCaml Lwt language extension, an event-driven asynchronous programming library, to enable a multi-threading programming abstraction. Using the Lwt syntax, a developer can spawn new threads by annotating closures as blocking. An Lwt thread is either executing code or idles waiting for an IO event or a synchronisation primitive (e.g. a semaphore) or a timer timeout. Lwt threads are cooperative and non-



(a) ns-3 (b) Xen

Figure 3.9: SDNSIM host internal architecture: ns-3 simulation 3.9(a) and xen real-time emulation 3.9(b).

preemptive and the main scheduling loop works as follow: Execute any resumable sleeping threads, calculate the time left until the next timer event and poll for events from the event channel until that time expires. Timing integration with the Xen platform is achieved through the Xen shared_info struct, a structure containing CPU time information which is shared between the Dum0 and the DumU VMs. Network integration is implemented using the Xen NetFront driver and the Xen event channel.

SDNSIM uses the Xen Managment API (XAPI) Xen Project to bootstrap emulations over the XEN platform. SDNSIM is able to create, start and stop VM instances and configure network topologies. Network topologies are implemented using VM Virtual Network Interface (VIF) bridging in the Dum0. A link between two hosts in an experiment definition is translate to a layer-2 bridge which contains a VIF from each VM. XAPI also expose an interface to control link rate and propagation delay for each VIF. In a Xen-based emulation of an SDNSIM experiment, the emulation setup pipeline works as follows: host definitions are compiled in VM images and equivalent VM definition are configured through XAPI, network topology description is translated into equivalent bridge setup and, once all configurations are completed, all VMs are booted.

3.7.2 ns-3 backend

ns-3 Henderson et al. [2006] is a discrete-time packet-driven network simulation framework. The core of the system consists of a discrete time simulation engine and a set

of ns-3 libraries providing network applications, routing protocols, data-link layer protocols and link emulations models. ns-3 is fundamentally an extensive refactor of the ns-2 code-base and provides a stable core of network functionality over which users can develop custom simulation scenarios.

The ns-3 backend has a significant difference from the Xen backend; the execution model is discrete and non-reentrant. ns-3 applications can only 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 the Lwt library to the ns-3 simulation engine, we transform 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 while the queue is full. Finally, in order to avoid scheduling deadlocks, the OS schedules *idle* time events that resume any yielded threads. Using these transformations we are able to provide a semantically accurate Lwt integration with the ns-3 engine. Network links between hosts are simulated using the *PointToPoint* link model. This model simulates a PPP link over a lossless medium, a valid approximations for the full duplex non-shared medium of current network datacenters.

Finally, in order to increase the scalability of the SDNSIM simulation backend, we use a distributed version of the ns-3 simulation engine. 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 execution Pelkey and Riley [2011].

3.8 SDNSIM evaluation

We evaluate performance and precision of the SDNSIM platform using small scale micro-benchmarks that target the performance of the OpenFlow protocol library, as well as, the scalability of the ns-3 backend. In Madhavapeddy et al. [2013], there is an exhaustive analysis of the performance of the Mirage platform, which we omit

from this section. In the rest of this section we compare the performance of our Mirage OpenFlow controller (Section 3.8.1) and Mirage OpenFlow switch (Section 3.8.2) with existing equivalent software packages and evaluate the scalability of the ns-3 backend (Section 3.8.3).

3.8.1 Mirage Controller Emulation

We benchmark our controller library's performance through a simple baseline comparison against two existing OpenFlow controllers, NOX and Maestro. NOX Gude et al. [2008b] 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 compare 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 compare 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 `packet_in` messages. The program measures the processing throughput of each controller. It provides two modes of operation, both measured in terms of `packet_in` requests processed per second: *latency*, where only a single `packet_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.

We emulate 16 switches concurrently connected to the controller, each serving 100 distinct MAC addresses. We run our experiments on a 16-core AMD server running Debian Wheezy with 40 GB of RAM and each controller configured to use a single thread of execution. We restrict 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 run the experiment for 120 seconds and measure the per-second rate of successful interactions. Table 3.5 reports the average and standard deviation of requests serviced per second.

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
Mirage Xen	98.5	4.4	24.5	0.0

Table 3.5: OpenFlow controller performance.

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 suffer 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 the performance of realistic OpenFlow deployment.

3.8.2 Mirage Switch

We benchmark our Mirage OpenFlow switch implementation through a baseline comparison with the Open vSwitch (OVS) Pettit et al. [2010] kernel implementation. We develop, using the OFLOPS framework, a simple forwarding test and measure the switching latency incurred by each implementation. For this experiment we use two virtual machines, one running the OFLOPS code, the other running the OpenFlow switch configured with three interfaces bridged separately in dom0. One interface provides a control channel for the switch, while the other two are used as the switch's data channels. Using OFLOPS, we generate packets on one of the data channels and receive traffic on the other, having bootstrapped appropriate the switch flow table. We run the

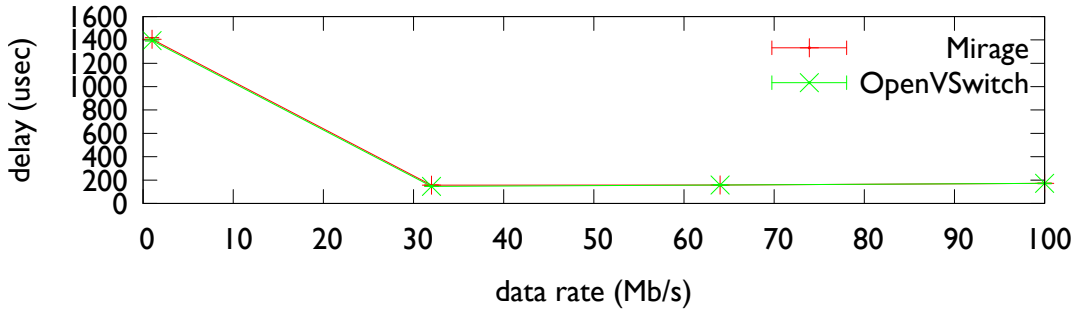
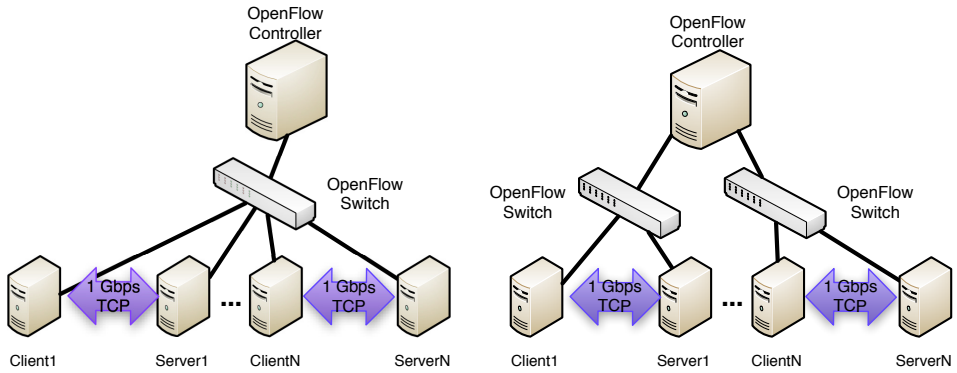


Figure 3.10: Min/max/median delay switching 100 byte packets when running the Mirage switch and Open vSwitch kernel module as domU virtual machines.

test for 30 seconds, a sufficient measurement period to detect statistically significant results. We use small packets (100 bytes)[1] and vary the data rate.

Figure 3.10 plots as error boxes the min, median and max 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 Mb/s; we believe this is due to a performance artefact of the underlying dom0 network stack. We omit packet loss, but can report that both implementations suffer similar levels of packet loss. From the comparison results of the Mirage OpenFlow switch, we can conclude, that our OpenFlow switch can emulate accurately software switch functionality. Nonetheless, the minimum latency of our switch emulation (approximately 100 msec) is 2 order of magnitude higher than a hardware switch (approximately 0.5-1 μsec). In order to bridge this latency gap, we propose the introduction of a slowdown factor in the emulation. A slowdown factor of ten will reduce by ten times the time reported by the clock abstraction of the Mirage platform and by ten the link rate allocated to each VIF. As a result, the slowdown factor can provide a semantically correct slowdown to the time of the experiment and use more efficiently the computational resources of the server.



(a) centralised topology (b) distributed topology

Figure 3.11: Network topology to test the scalability of ns-3-based simulation. We simulate two topologies: a centralised topology (Figure 3.11(a)) and a distributed topology (Figure 3.11(b))

	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 a 30 seconds simulation time of the topology presented in Figure 3.11, for variable number of network hosts. SDNSIM simulation of a network experiment using the ns-3 backend scales linearly as the traffic is localised.

3.8.3 ns-3 performance

In order to test the scalability of SDNSIM simulation we implement a simple network experiment, depicted in Figure 3.11. The topology consists of a number of switches and host pairs, connected through 1 Gbps links. Each pair of hosts generates steady state TCP traffic at line rate. We use two variations of the topology: A *centralised topology* with all the pairs of hosts connected to a single switch (Figure 3.11(a)), and a *distributed topology*, with pairs of hosts distributed between two switches and traffic remaining local to the switch (Figure 3.11(b)). Each switch is connected to an OpenFlow controller that implements a learning switch. The experiment executes 30

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

Parameter	value
packet_in processing delay	10 msec
flow_mod insertion delay	1,2 msec
link capacity	1 Gbps
Request arrival rate	1500 retrievals/sec
link propagation delay	0.5 μsec
switch processing delay	100 μsec

Table 3.7: control hierarchy simulation parameters

seconds of simulation time on a 24-core AMD server with 128 GB of RAM.

In Table 3.6, we present 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. Additional, 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 parallelized. In the distributed topology, network events are distributed between the two switch and they execute independently, achieving near-linear scaling. These scaling properties of the SDNSIM simulation implementation provide a scalable experimentation framework for datacenter traffic models and designs Kandula et al. [2009].

3.9 hierarchical control distribution in Software-defined network

The OpenFlow abstraction provides an excellent mechanism to centralise control in a network and avoid the performance overhead of existing distributed control plane protocols. Using OpenFlow, allows to offload the control plane of all switches of a network to a single centralised network controller which has full knowledge of the network state and construct an optimal routing policy. Nonetheless, as the network size increases, the centralisation strategy will face significant problem to scale centrally the control plane requirements. Large networks require a mechanism to distribute the

control computation over multiple tightly-coupled controllers. In order to address this requirement we use SDNSIM to test a hierarchical control system. In our control architecture, we propose the introduction of a hierarchy of controllers which control a subset of the switch devices and a subspace of the OpenFlow tuple space. In order to enable control distribution, as well as, responsive control, each node of the hierarchy run a flowvisor instance to multiplex switch control channels. The flowvisor is configurated to delegate control to a local controller, while also propagating control events to upper layers of the hierarchy in order to build higher level controllers which construct global view of the network and enforce higher priority control policies. In order to understand the impact of this control hierarchy, we use SDNSIM, to simulate the control architecture and evaluate the impact of multi-level control schemes on the performance of the data plane.

In our experiment we use a FatTree Al-Fares et al. [2008] network topology with 4 pods and simulate a small-scale datacenter. For data plane traffic we use a simple client-server architecture to replicate the functionality of data retrievals in Map-Reduced platforms. In the network we assign the machines hosted in the first three pods to function as data requesters and the machines hosted in the forth pod to function as data providers. Each data request consists of a small 10 bytes packet from the requester, to define the number of bytes which will be send subsequently by the data provider. The traffic size of the request is selected uniformly at random between the values 4KB, 8KB and 32KB. The request arrival rate follows a Poisson model, while the destination is choose uniformly at random between the servers of the forth pod. In addition, each client keeps a persistent long-lived background TCP connection. Using this simple model we can experiment with traffic models similar to a datacenter network. In the control of the network we simulate the control logic of the FatTree design: every 1 second the network control receives flow-level statistics from the switch and reassign network flows in order to reduce link utilisation and spread the load between redundant. In order to study the effects of our architectural approach, we variate the number of FlowVisor instances which process OpenFlow traffic before reaching the local controller of the switch and study the impact of the proximity of the controller on data plane performance, by measuring the completion time of each network flow. Further parameters of the simulation are described in Table 3.9.

3.10 Summary

This chapter discusses the scalability capabilities 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 use OFLOPS to evaluate five different OpenFlow switch implementations, in terms of OpenFlow protocol support as well as performance. In our performance evaluation, we benchmark the packet processing, traffic interception and injection, flow table modification and traffic statistics export functionalities of the switches. We identify considerable variation among the tested OpenFlow implementations. We take 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. Further, at high rates, monitoring operations impact other OpenFlow commands.

In addition, in this chapter we present the SDNSIM platform, an experimentation framework for large-scale high-precision experimentation with control plane architectures. SDNSIM reuses the Mirage systems abstraction layer to enable developer to program the functionality of their network. An SDNSIM experiment definition can seamlessly translate at compile into a simulation over the ns-3 simulation platform, or a high precision emulation over the Xen virtualization framework. We conduct a number of micro-benchmarks and evaluate that the software provide emulation performance comparable to software OpenFlow controller and switching platforms, while SDNSIM simulation provide good scaling properties. Furthermore, using SDNSIM we 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.

More specifically, in this chapter we present the engineering and social characteristics of home networks (Section 4.1) and elaborate on the nature of the problems and the inherent opportunities of the specific network environment (4.2). Furthermore, we describe our home router design (Section 4.3) and present and evaluate a number of proposed protocol modifications placing the homeowner in direct control of their network (Section 4.4) . In Addition, we present a simple QoS mechanism enabling user-ISP collaboration to improve resource utilisation in the last-mile using commodity SDN devices (Section 4.5). Finally we conclude the results of our exploration (Section 4.6) .

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 Technological and Social aspects of home networking

The growth in the number of IP-enabled devices over the last decade has expanded the use-cases for in-home wired and wireless networking from Internet connection sharing to local media sharing, gaming, and other applications. Home network functionality, unlike other network environments, contains a strong social aspect, which establishes a dynamic relationship between the technology and the social organisation of the setting and thus exhibits unique properties. In this Section, we elaborate results from various home network studies and elaborate on the characteristics of home network environments, both from a social, as well as, an engineering perspective. More specifically, in Subsection 4.1.1 we present the distinct technical characteristics of home network technologies, while in Subsection 4.1.2 we focus on the social characteristics of home networking, using findings from relevant ethnographic studies.

4.1.1 Home Networking as a system

Home networks are highly heterogeneous edge networks, typically Internet-connected via a single broadband link, where non-expert network operators provide a wide range of services to a small set of users. Home networks use predominantly ADSL and cable technologies for Internet connectivity, while fiber, 3G and satellite technologies are not uncommon. While we focus on home networks, we note that many environments, e.g., small offices, coffee shops, hotels, exhibit similar characteristics and thus may benefit from similar approaches. Such capabilities are likely to be infeasible in more traditional settings, e.g., backbone and enterprise networks. In this section we use existing studies to present the technical characteristics of modern home network.

Home networks commonly provide local device connectivity through wired and wireless Ethernet, both supported by modern PCs. Wired network adapters predomi-

nantly support the 802.3ab (1 Gbps full-duplex) standard and wireless network adapters support predominantly 802.11a/g (54 Mbps) standards with hardware-accelerated encryption, while 802.11n (600 Mbps) support slowly becomes commodity. Wired Ethernet connectivity provides high data rates with negligible medium interferences, but fails in supporting flexible spatial mobility within the household. In contrast, wireless Ethernet connectivity supports extensive device mobility, but network performance is susceptible to environmental parameters. A number of studies have tried to study the behaviour of wireless Ethernet in an effort to improve performance. In Cioccio et al. [2011] the authors develop the Homenet measurement framework, an end-host active measurement tool integrated with an active user survey. Results from the Homenet measurements tool report on average good signal strengths for wireless home networks ($<$ -80dBm), while the measurement tool picks up advertisement, on average, of 10 other wireless networks, a third of which used overlapping channels. These observations point out the continuous improvement of wireless technology and contrast earlier results in Yarvis et al. [2005]. In addition, the visibility of wireless networks beyond the physical limits of the home highlights a significant access control problem. The WPA encryption scheme ensures network data integrity but the control scheme is inflexible; pre-shared key knowledge provides full access to the home network.

Furthermore, home Internet connectivity exhibits variable performance, dand depends to a great extend on the ISP configuration. In Dischinger et al. [2007a], the authors contact an active network measurement of residential broadband ISPs and identify significant performance variance. In addition, the study identifies a performance bottleneck on the last mile of the broadband link. In a relevant study, the authors in Sundaresan and de Donato [2011] contact a long-term study of broadband connectivity, using the home router to run various networks test, and detect significant effects on network throughput and latency by ISP throttling mechanisms, like PowerBoost DSLreports [2013]. In their analysis they conclude that broadband performance can be affected by multiple factors, distributed across the network, while accurate performance characterisation cannot be contacted through a single measurement test. The effects of the ISP network configuration in Internet connectivity are measured by the Netalyzer Kreibich et al. [2010] tool, a network connectivity and protocol openness evaluation framework. Netalyzer uses a network buffer discovery test which measured up to 200 msec of buffer latencies in popular ISP networks. Such Buffering latencies

are common to develop during pick hours and affect to a great extend the user experience. The importance of Internet performance for residential broadband networks is reflected on a governmental level, where independent state authorities evaluate ISP service quality Federal Communications Commission [2012]; ofcom [2012].

Home networks exhibit interesting characteristics with respect to applications and traffic properties. Home network commonly have relaxed network policies, predominantly defined and enforced by the ISP, and, as a result, permit usage of a wide range of applications. In Reggani et al. [2012] the authors develop a passive measurement tool, logging continuously end-host traffic. Trace analysis unveils that in a home network setting end-hosts generate and consume significant volumes of network filesystem and P2P applications, while a large portion of the traffic remains local and is observable only from within the network. Residential broadband Internet traffic patterns exhibit significant temporal and spatial variability. In Cho et al. [2006] authors detect in 2005 a significant volumes of P2P traffic in Japanese bradband ISPs. More recent studies Maier et al. [2009] of Internet traffic from a German ADSL provider point out a shift of user trend towards web applications, with a large portion of traffic being HTTP-based. Similar trends are detected in USA, as presented in Erman et al. [2011]. In this study the authors analyse full packet traces from an ADSL provider and identify that 70% of the network traffic uses the HTTP protocol, primarily for online video and one-click hosting services.

Finally, home network studies identify an increase in the number of devices connected to a home network, as well as, significant heterogeneity between the devices. In the Homenet Cioccio et al. [2011] survey users report, on average, 7-8 network enabled-devices installed in the local network, while active probing of the network discovered, on average, only 1-2 active devices. Further, the user survey reports a wide range of connected devices (e.g. smartphones, tablets, game consoles, etc.). This points out that any change of the home network functionality must be backwards compatible with a wide range of operating systems and devices. Nonetheless, in a similar study in Hätönen et al. [2010] the authors test a number of off-the-self home routers and unveil high variability in the semantics of the NAT and DNS functionality implemented by the router firmware. Since such routers are widely deployed in home networks, device network functionality, as well as, users are accustomed to such variability.

4.1.2 Home Network as a social activity

Home networks technologies have been a domain of interest for the HCI and sociology research communities, as they provide an interesting environment to study the interactions of social behaviours and technology. Studies in the field usually engage in user interviews in order to understand how users perceive and interact with network technologies.

An important aspect in analysing the home network from a social aspect, requires an in-depth understanding of the user perception of home network technologies. In Grinter and Edwards [2005]; Shehan-Poole et al. [2008] the authors use sketching to enable users to describe their understanding of home network connectivity. The analysis highlights two important observations; user opacity to the network increases inversely proportional to their user network experience – verifying the effectiveness of the abstraction-based design of the current network stack, and users identify network devices using the social context of the home network. Further, in Tolmie et al. [2007] the authors perform an empirical analysis of the interactions of home members with network technologies. Interestingly, their finding identify that, on average, users are least motivated to optimize home network configuration, as long as the perceived performance is tolerable. Additionally, users interest to engage in network maintenance increases when the task resembles other common household duties (well defined and simple tasks with short durations), while the interest conflict that arise due to the shared nature of such infrastructures is usually solved through negotiation between the household members. Relevant studies have highlighted the importance of accurate information of the utilization and state of the network for network management. In Chetty et al. [2010], the authors develop a visualization system, informing home network users with statistics on bandwidth utilization. Interestingly, the system improved user understanding of network functionality and troubleshooting, but, additionally, raised privacy concerns.

Home network configuration and management are influenced to a great extend by the design of the house and the routines of the house members. In Rodden and Benford [2003] authors use a model of home architecture temporal evolution Brand [1995] to analyse the relationships between ubiquitous technologies and home design and understand the interactions between home design and network usage. The study describes

how the user network decisions are affected by the design of the house, e.g., location of the network router, while at the same time how the users confuse the limits of the house with the limits of their network, e.g. users assume that encrypting their home network is not important since it is contained within the limits of the house. In Shehan and Edwards [2007] authors analyse common management and configuration problems in home networks and project them in the respective design decisions of the network systems. They 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.

add reference to Mazurek et al. [2010] for access control requirements for the house.

4.2 Motivations

4.2.1 Home Network: Use cases

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. Its 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 Shehan and Edwards [2007]; Shehan-Poole et al. [2008]; 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 technology 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.*

Sharing the medium socially efficient. *Richard is the teenage son of Derek and has a great interest in Music, downloading a lot of music from the Internet. Derek works some times in the night from home using the Terminal Services provided by his employee. Commonly Tension is created between the household member, 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.

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.

4.2.2 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.

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;

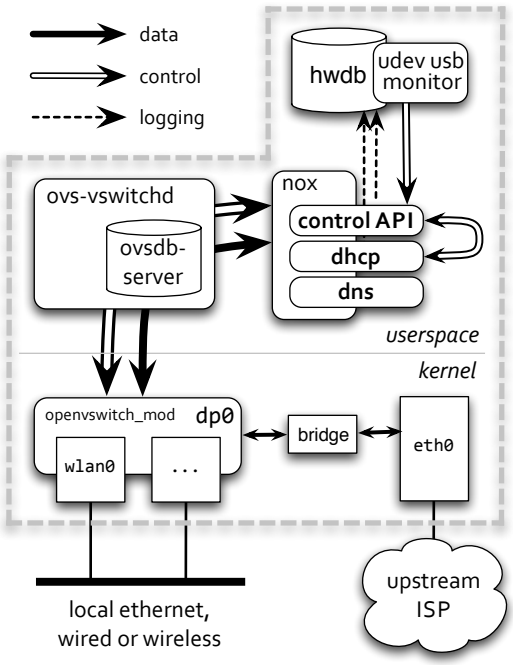


Figure 4.1: Home router architecture. Open vSwitch (*ovs**) and NOX manage the wireless interface. Three NOX modules provide a web services control API, a DHCP server with custom address allocation and lease management, and a DNS interceptor, all logging to the Homework Database (*hwdb*) (§4.4).

- 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.

4.3 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

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

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, we provide a number of novel user interfaces, one of which we describe 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.

4.3.1 OpenFlow, Open vSwitch & NOX

We provide OpenFlow support using Open vSwitch Pettit et al. [2010], OpenFlow-enabled switching software that replaces the in-kernel Linux bridging functionality able to operate as a standard Ethernet switch as well as providing full support for the OpenFlow protocol. We use the NOX Gude et al. [2008b] controller as it provides a programmable platform abstracting OpenFlow interaction to events with associated callbacks, exporting APIs for C++ and Python.

Our network control logic, discussed in Section 4.4, is implemented using the NOX controller framework through five different modules. The C++ module *hwdb* synchronizes router state with the hwdb home database, presented in Section 4.3.2, 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 the DNS interception functionality and the Python module *homework_rpc* exposes the control API as 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.[1] Control of the router 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.[2]

[1] Without loss of generality, our home router has only a single wired interface so the only home-facing interface is its wireless interface; other home-facing interfaces would also become part of the OpenFlow datapath.

[2] While NAT functionality could be implemented within NOX, it seemed neither interesting nor

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.

4.3.2 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, 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

necessary to do so.

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

performance; and *Leases*, recording mappings assigned via DHCP.

4.3.3 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. We exploit this physical arrangement to provide a focal point for inhabitants to view current network status and to manage the network. It 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 4.4. 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.

4.4 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.

4.4.1 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 nego-

tiate 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 (30s) 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 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 0.7s.

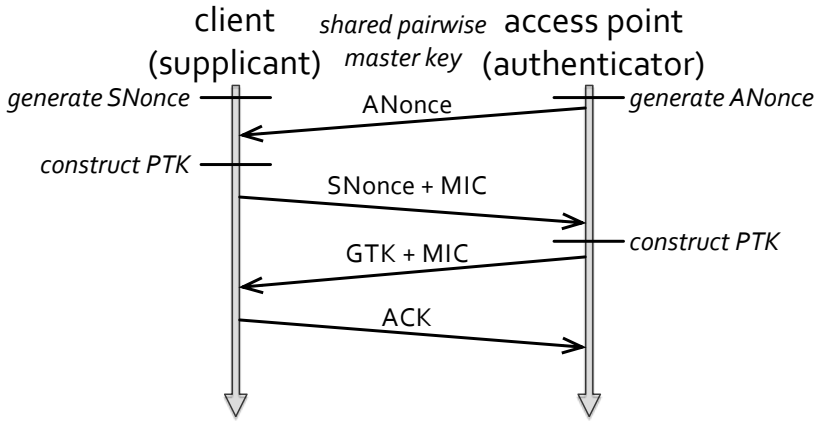


Figure 4.3: 802.11i handshake, part of the association process. Note that MIC (Message Integrity Code) is an alternate term for MAC, used in such contexts to avoid confusion with Media Access Control.

4.4.2 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 on the same IP subnet.[1]

Periodically, a timeout event at the access point initiates rekeying of the PTK, vis-

[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.

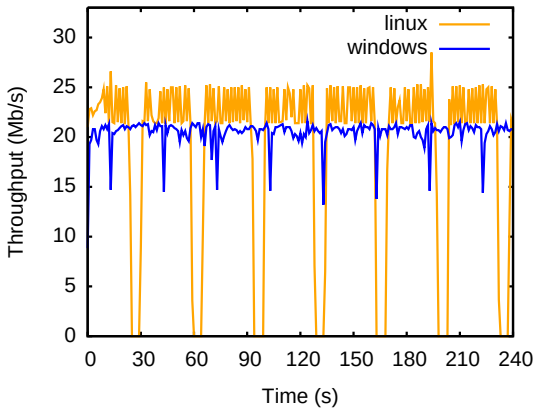
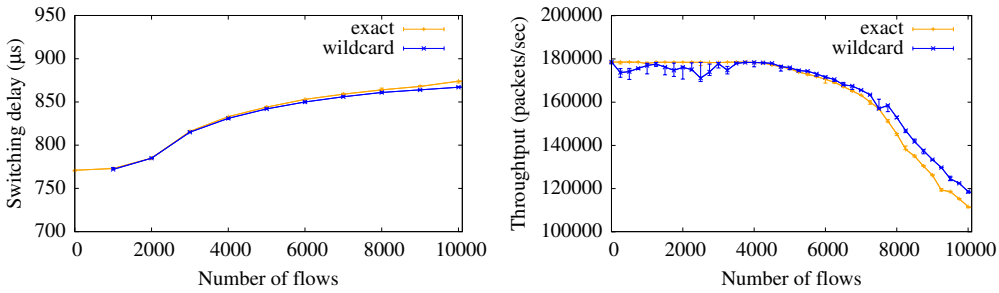


Figure 4.4: Affect on TCP throughput from rekeying every 30s for Linux 2.6.35 using a Broadcom card with the *athk9* module; and Windows 7 using a proprietary Intel driver and card.

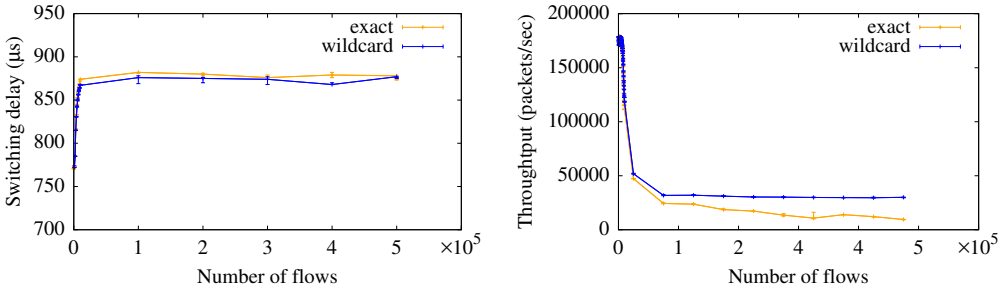
ible to clients only as a momentary drop in performance rather than the interface itself going down. We use this to apply blacklisting of clients deemed malicious, such as a client that attempts to communicate directly (at the link-layer) with another client, i.e., attempting to avoid their traffic being 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 30s: rekeying causes a periodic dip in throughput as the wireless Ethernet transparently buffers packets during rekeying before 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



(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.

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.

4.4.3 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 30s idle timeouts, ensuring that they are garbage collected if the flow goes idle for over 30s.

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 oberve the network performance behaviour, generating packets at line rate from a 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 (1500B) for the latency tests and minimal packets (70B) [2] 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.

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

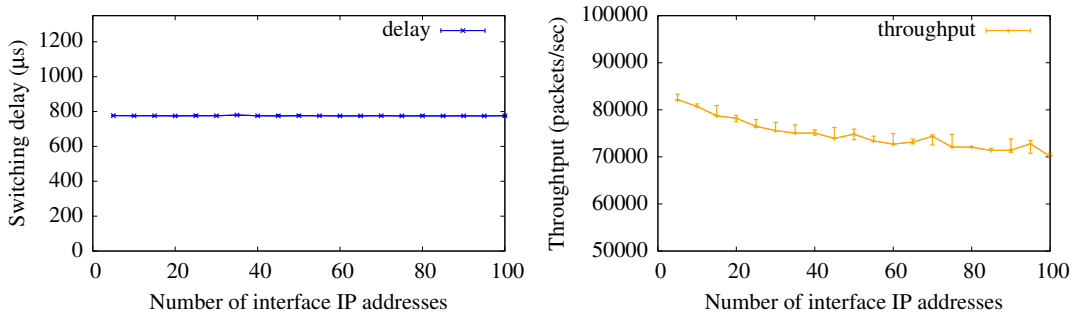
[2] The 30B extra overhead is due to *pktgen* Olsson [2005a], the traffic generation tool used.

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 560μs and minimum throughput of 40,000 packets/second (Figure 4.5(c),4.5(a)); initial deployment data suggests a working maximum of 3000 installed flows which would give around 160,000 packets/second throughput (small packets) and 500μs 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 thr Linux kernel to implement longest prefix match, which scale lookup complexity logarithmically with respect to the number of IP addresses.

4.4.4 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 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.



(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.

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.

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

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 30s.
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.

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. 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

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

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 recent 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 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 of our approach, we set up a simple hardcoded 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 allo-

[1] OpenFlow provides support in its 1.4 release of the protocol; NOX currently has no support for IPv6; 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.

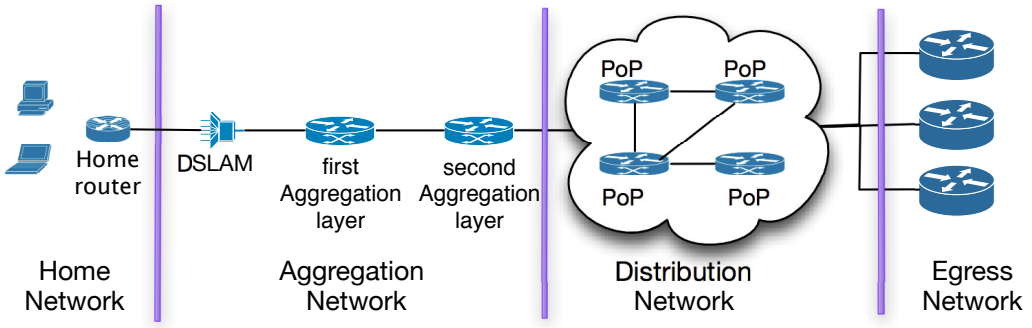


Figure 4.7: The path for each network packet of the home network to the Internet. ISP network can be split in 3 parts: The *Aggregation Network*, the *Distribution Network* and the *Egress Network*.

cate 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 test MacOSX, Windows and Linux IPv6 network stacks and verified correct network configuration and Internet connectivity.

4.5 User-centric resource allocation

As we have discussed in Section 1.1.1, network application evolution has increased both the per-host resource requirements, as well as, the complexity of the resource allocation mechanism. Effectively, developing network architectures that can provide global application-specific network performance guarantees is a difficult task. The network community uses two common practices to tackle the resource allocation problem: network resource over-provision and application-aware traffic engineering. Resource over-provision increases available resources, in order to remove performance bottlenecks. Application-aware network engineering extends the network policy to manage efficiently traffic from specific applications, e.g. in an enterprise network VoIP devices are connected to a separate high-priority VLAN. In broadband networks these approaches have limited applicability. On one hand, resource over-provision cost is non-scalable on the edges of the broadband network [1], while, on the other hand, application-

[1] optical link installation costs varies between $50-$200 per foot, while fibers installation in municipal areas has an annual cost of $1.91 per foot News [2011]. Upgrading such links also has a significant

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, which enables homeowner to propagate per-application resource requirements. Using this mechanism, we scale the ISP resource allocation policy, while users can improve their network experience and understand performance limitations. In the rest of this section, we present the architecture of current broadband networks and discuss the inherent limitations in resource allocation (Section 4.5.1), we present a highly efficient network architecture for user-driven resource allocation (Section 4.5.2) and evaluate the impact of our approach on the performance of various traffic types (Section 4.5.3).

4.5.1 ISP resource allocation

In order to better introduce the reader to our system, we present in Figure 4.7 a visualisation of a broadband ISP network architecture. Although, exact network architecture remains undisclosed by ISPs, there is a high level design pattern. In this figure we present the network devices traversed by a packet sent from the home network to the Internet. The ISP network organisation can be separated in four sections: the *home network*, *aggregation network*, *distribution network* and *egress network*. The aggregation network contains the digital subscriber line access multiplexer (DSLAM), translating DSL packets to Ethernet format between the last mile, the Broadband Remote Access Server (BRAS), an authenticating and policy enforcing network device, and a series of aggregation switches, multiplexing the traffic from the various BRAS onto the distribution network. Traffic from the aggregate network is routed through the distribution network towards either the egress network of the ISP or towards a local aggregation switch. The distribution network consists of a small number of large routers interconnected using high capacity links. ISP typically employ network tunnelling technologies, like MPLS, in the distribution network, in an effort to minimize forwarding state in devices. The egress network of the ISP consist of large routers, hosted in IXP networks and connecting the ISP with peering ASes.

In the broadband network architecture, a number of research papers have identified a significant bottleneck in the last-mile of the network Akella et al. [2003]; Dischinger et al. [2007b]: the link between the DSLAM and the aggregation network. This bot-

network equipment upgrade cost.

tleneck can be explained by the high oversubscription ratio of such links, which commonly varies between 10:1 and 50:1 Program [2013], depending on the number of users connected to the DSLAM and market dynamics Leach [2013].

Our network design aims to bridge a fundamental communication gap between the ISP and homeowner. Users perceive network traffic as an ensemble of flows belonging to a specific application and each application has a specific prioritisation within the house context and the user-computer interaction, 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, on the other hand, manage the traffic of a specific household as a packet aggregate with a specific source IP address and resource allocation mechanisms are implemented as monthly or daily traffic caps for the specific IP address British Telecoms [2013]; Virgin Media. Network caps rate-limit heavy-hitting households and provide long-term fairness between broadband users. Nonetheless, this approach is not sufficient for a class of Internet applications, like remote desktop and VoIP, for which performance is not tightly coupled with the available bandwidth. We believe that there is an incentive for the ISP and homeowner entities to establish a collaborative resource control framework. Homeowner can annotate high priority flows within their traffic aggregate and thus improve user experience. ISPs can increase user-friendliness of their network policy and offload some of the complexity of their resource allocation mechanism to the end-users.

Our architecture establishes a communication channel between the household and the ISP and enables user-driven dynamic resource control on the bottleneck link towards the aggregation network. More specifically, the ISP virtualises switch resources over the last-mile and delegates control over a subset of the available resources to a homeowner-managed OpenFlow controller. The household controller, in real time, uses the OpenFlow protocol to assign traffic to the 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 congestion on the edge-network and scale the configuration complexity.

4.5.2 User - ISP communication

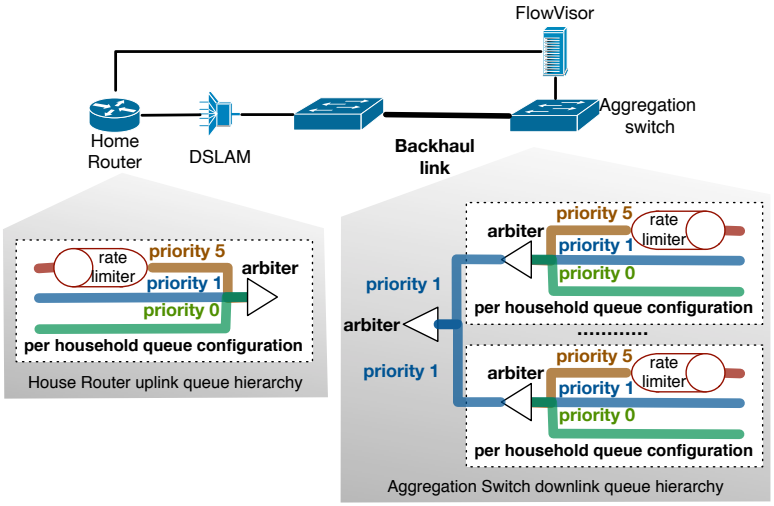


Figure 4.8: User-driven network resource allocation architecture. Contrl over the downlink between the backhaul network and the DSLAM is exposed through a virtual slice to the homeowner through a FlowVisor instance. The switch allocates three queue primitives per-household: A *low latency, high priority* queue, a *medium priority* queue and a *default* queue.

Our architecture consists of three distinct subsystems: a data-collection daemon on the end-systems of the home network, a policy enforcing daemon on the home network router and an OpenFlow controller trnaslating user performance requirements into ISP control policy.

End-system In order to map network flows to applications, we develop a light cross-platform daemon recording the content of the end-host connection table. Information are collected from the connection table of the network stack 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 applications, we use the `lsof` command in unix-like systems and the `netstat -p` command in Windows.

ISP infrastructure We present the proposed edge network architecture of the ISP network in Figure 4.8. Our design relies on an OpenFlow-based control mechanism on

the the first layer of aggregation switches. The switch control plane is virtualised using the FlowVisor controller Sherwood et al. [2010b]. Each home router is running an OpenFlow controller which connects with the FlowVisor service and exercises control over the home Internet traffic. Specifically, the ISP virtualises the OpenFlow forwarding table and exposes only a subset of the entries to each user. The FlowVisor instance propagates `pkt_in` messages to the home controller with a source or destination IP address equal to the home public IP address, and discards *flow_mod* messages without an IP address equal to the household address in the source or destination address. Such FlowVisor configuration enforces a strong security mechanism; home controllers cannot control or eavesdrop traffic from other households.

At the aggregation switch and the home router we setup three priority queues for each household (The home router queues handle the upstream traffic of the household, while the aggregation switch queues handle the downstream traffic of the 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 uses a leaky bucket mechanism to rate limit traffic to the minimum guaranteed bandwidth per-household. This queue can be used by low-bandwidth low-latency applications. The HP queue doesn't enforce any rate limit, but it has a priority which is between the LLHP and the default queue. HP 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 an Round-Robin scheduler with equal priority for each household. Large scale queue hierarchies are currently supported by service provider grade network equipment (e.g. Cisco ASR 9000 router provide 500k 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 run-time can assign flows to higher priority queues based on the user policy. This resource allocation mechanism is constrained per-household by the number of flows that the FlowVisor exposes to each home. This network design can be extended to evolve further the ISP broadband economical model; Users can enhance network performance by purchasing additional flow table entries and increase minimum guaranteed bandwidth on the edge.

Because our system design provides extensive forwarding control to end-users,

we fortify the design with a set of mechanisms to reduce the ability of end-users to compromise network functionality. 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_mod` messages with an Output action that forwards packet to the incoming port.

Home Router In our design, the home router is the rendez-vous point between the policies of the two network. In order to support the state requirements of this functionality we create in the hwdb database three tables: *Application tuple*, *Application timeseries* and *Application priorities*. Application tuple table stores mappings between applications and network tuples. This table is populated with data from the end-hosts, while the router installs a monitoring hook in order to receive new flow arrivals. Application timeserries table stores per-application network utilization information. The table is populated with data from the router using the `flow_stats` message of the OpenFlow protocol, while the data are used by the web visualisation in order to inform users for the per-application network resource consumption. Finally, the Application priority table stores the application prioritisation policy of the user. This table is populated with data through the web interface of the system and the controller uses it to map new flows to appropriate queues.

We expose the resource management mechanism through a user-friendly web interface depicted in Figure 4.9. Through this interface the user can view the aggregate and time serries resource consumption per application and configure application priority policy. Additionally, the interface notifies the user when the policy over-utilizes the LLHP queue. Specifically, we use the packet loss counter of the OpenFlow `queue_stats` message to trigger notifications during high packet loss incidents. Through this interface we try to address the issues raised by the work in Chetty et al. [2010].

During operation, the proposed design extends the forwarding logic described earlier. Specifically, for each network flow, once the controller has verified that the flow is in accordance with the policy and destined to a non-local IP address, the controller maps the flow to an application and to a priority queue, through the HWDB database.

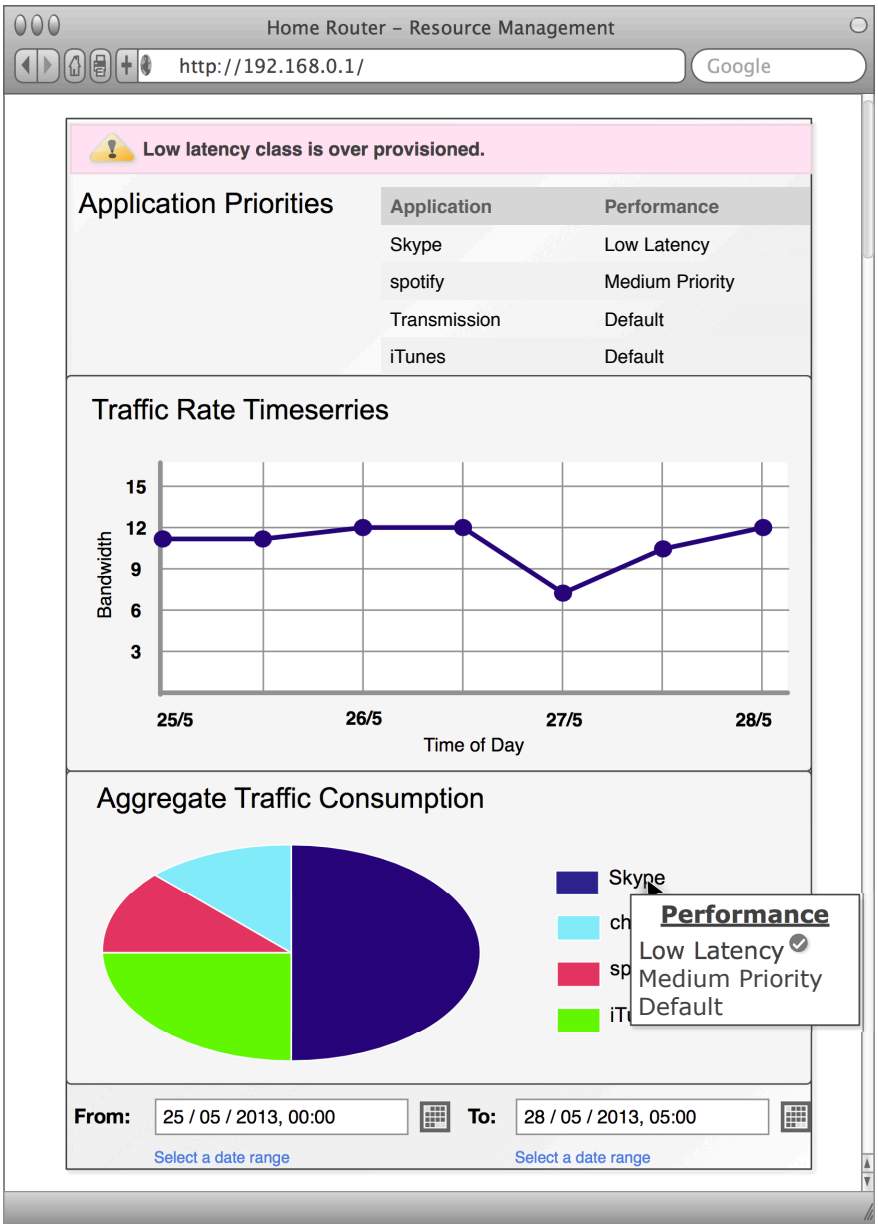


Figure 4.9: Performance mechanism user control. The user is able to view the network utilisation per application, as well as express application prioritisation.

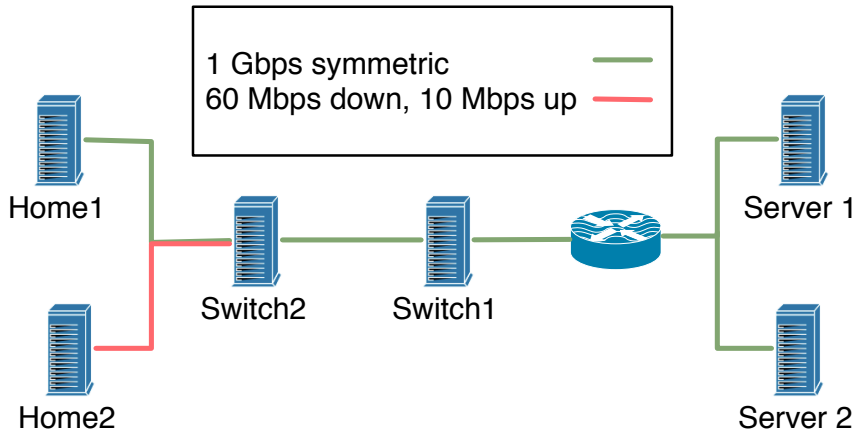


Figure 4.10: Experimental setup to evaluate the proposed resource management system.

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.

4.5.3 Evaluation

We evaluate the proposed architecture using a lab testbed, depicted in Figure 4.10. *Home1* and *Home2* emulate the networking activity of an ensemble of Home Connections. We focus the analysis on the behaviour of Home2, that emulates a single connection in the vicinity of an ensemble of Home connections, emulated by node Home1. *Server1* and *Server2* instances run a number of listening daemon that generate traffic, based on traffic requests, from the Home instances and simulates Internet wide Services. Fnally, *Switch1* and *Switch2* emulate the switches that control the backhaul link. Steady state TCP is generated using the iperf tool Tirumala et al. [2005]. We also utilize pttcp tool, in order to simulate stochastic models of short lived TCP flows.

4.6 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. Two partic-

ular user needs that arose from home network user studies were for richer visibility 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 to shape how it is presented and opened to user control and ultimately provide scalable management through a novel control plane architecture.

We use the Open vSwitch and NOX platforms to provide flow-based management of the home network. As part of this flow-based management, we exploit 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 only *not* their wire formats, due to the need to retain compatibility with legacy deployed stacks. In addition, in our exploration we present 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 connectivity scalability.

Chapter 5

Signpost: Internet-scale secure and scalable connectivity

In this chapter we explore applications of the SDN technology in connectivity scalability. The work is motivated by a common network use-case, seamless information and resources sharing between the devices of a user. We call this functionality Personal Cloud. Currently, Personal Clouds functionality employs Cloud services as information caches, bridging the inter-device communication requirements. This architecture is a consequence of the connectivity scalability limitations experienced by network applications, 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 the network connectivity limitations and regain control of their data by establishing user-managed Personal Cloud. We propose Signpost, an evolved network control plane for end-devices, delivering continuous and secure connectivity between the devices of a user. Signpost integrates existing network testing approaches to detect network environment conditions and configures existing software packages to establish end-to-end connectivity between devices.

In order to understand the feasibility and performance of the proposed architecture, we present a strawman implementation of the core control logic of the proposed architecture and its integration with a number of network connection and notifica-

tion services. Currently, Signpost supports SSH, OpenVPN, TOR, NAT-punch and Privoxy connection 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 observation, we define the key design requirements for our platform (Section 5.1). Furthermore, we present the Signpost design (Section 5.2) and architecture (Section 5.3), followed by an elaborate description of a strawman implementation of the system and an evaluation of its functionality (Section 5.4). Finally, we conclude our observations (Section 5.5).

5.1 Personal Clouds

In the recent years, the global increase in the adoption of personal computers has mirrored a large part of our life in the digital domain. For example, human labour is commonly stored in digital PDF and word processing files, while digital photo collections express a part of our social interactions. As a result, our physical presence has an increasing digital counterpart. A user interacts primarily with his digital presence through his computing devices. Although, the recent increase in the number of computing devices per-user introduces new challenges in the management of our digital presence. Complimentary to their personal computer, users on average currently own and use a tablet, a smartphone and a variable number of purpose-build computing units for home entertainment and gaming. Each user device fulfils specific roles in managing the user digital resources, thus requiring access only to a subset of the user digital presence, but these roles are fluid and intercept. 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. This transition from the single PC paradigm to the multi-device paradigm and the concurrent digital footprint augmentation create a user requirement for new services providing global and homogeneous access and control of personal digital information and resources. We call this abstraction the user *Personal Cloud*.

In the rest of this Section, we present existing mechanisms providing Personal Cloud functionality (Subsection 5.1.1) and discuss their functional limitations in order

to define the design goals for our Signpost system (Subsection 5.1.2).

5.1.1 Approaches

Currently, partial Personal Cloud functionality is readily available in a number of applications. Such applications provide a wide range of abstractions and each has different network resource and connectivity requirements. We group these approach in two distinct categories, based on the service architecture.

Decentralised Personal Cloud The requirement to share information and computing resources has been a core application for computer networks since their introduction. Currently the average user has access to a large number of free software packages and protocols that provide Personal Cloud functionalities, such as remote desktop access (e.g. Remote Frame Buffer protocol Richardson and Levine [2011]), file sharing (e.g. NFS Shepler et al. [2003], 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]) etc. between devices. Such applications follow the client-server connection paradigm and extend the computer abstraction to accommodate inter-device connectivity and resource accessibility. A user can setup and manage the equivalent services on his personal devices and enhance his user experience with Personal Cloud functionality. In addition, the network community has developed mechanisms to ease the configuration of such services. Multicast-DNS, UPnP and ZeroConf protocols allow a user to seamlessly browse and access available services upon connection to a network.

Decentralised Personal Cloud applications enable users to fully control their digital footprint and provide strong privacy guarantees. Data are stored on users devices and the user has full control on the propagation of his information. Nonetheless, deploying such service in the Internet raises significant scalability issues. We focus on three significant problems in Internet-scale deployments.

- *Connectivity*: Current Internet architecture cannot guarantee globally accessible services for all hosts, as we have discussed in Section 1.1, and as a consequence the effectiveness of the Client-server model of decentralised Personal Cloud applications is limited. End-users commonly connect to the Internet through edge

networks optimised for outgoing connectivity and transparent middleboxes commonly violate the Rubstness principle of the Internet. For example, NATed networks, that scale addressing on the edges of the network, limit incoming Internet connectivity for a significant number of devices and require manual configuration to expose services. Similarly, mobile and enterprise networks employ firewalls to restrict publicly accessible services on connected hosts for security reasons.

- *Authentication*: Distributed Personal Cloud applications employ an inflexible authentication mechanism, based on pre-shared passwords and public keys. Access to a decentralised service requires a secure channel to negotiate a-priori the security credentials for a user, while access control is integrated in the service configuration.

- *Usability*: Decentralised Personal Cloud applications introduce configuration requirements which dim them inappropriate for inexperienced users. As we have already discussed in Section 4.1.2, the average end-user is highly unlike to engage in systematic service configuration of decentralised services, since the configuration tasks are long and complex. Decentralised Personal Cloud services contain network functionality assumptions and users may need additional software layers, like tunneling software, to achieve functionality. In addition, as network policies exhibit variability, a user must resolve to "trial and error" approaches to evaluate which end-host configuration is effective in a specific environment.

  As an example, we describe the steps required to access files from a remote device using the SSH service. Before the user is able to connect to a computer over SSH, he has to configure the service on his server machine, both in terms of access control and service functionality, as well as, any firewall and NATbox services deployed in the local network. Additionally, an IP resolution service is required 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 over the preconfigured port. In case 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 cloud-based services to bridge information flows between devices. In this group we consider applications like the Google service suite, Dropbox file sharing service, LogMeIn Remote Desktop service and other alike services. These applications provide a simple, intuitive and ubiquitous mechanism to store and retrieve data between the devices of the users or a specific subset of users from the service extensive user-base, providing the ability to extend the sharing capabilities with a user social circle. For example, Google Docs allow users to store online documents and process them in parallel with a defined set of online friend. Although this approach provides all required functionality, some of design points of the architecture are ambivalent and demotivate user engagement.

- *Identity*: Cloud applications provide an effective control framework to disseminate information between devices and users. Users define information access policies based on online identities and the service ensures secure delivery. Nonetheless, as a service becomes popular and its functionality is integrated with third-party applications, the service provider gains the ability to identify users to third parties or monitor registered users. In Krishnamurthy and Wills [2009] the authors report a case of identity exposure by Online Social Networks to advertising services, in an effort to detect and characterize user interests. Facebook has openly verified the existence of such services Juan Carlos Perez [2007].

- *Performance*: The elasticity of cloud computing infrastructures enables centralise Personal Cloud services to provide scalable performance, irrespective of data volumes and service popularity. Nonetheless, the service architecture underutilizes the wealth of computational and network resources available in end-user devices. Two devices connected to the same subnet will experience bloated RTT values when they communicate through a cloud service, while the cost and performance of cloud storage resources is orders of magnitude higher in comparison to the cost of local storage. In Wittie et al. [2010], authors report a significant impact on the 95th quantile network performance of cloud services, due to latency and packet losses incurred by the Internet, while in Dean and Barroso [2013] the authors present a wide range of parameters, spanning from the CPU scheduler to hardware design, which influence user-perceived performance.

- *cost*: A large number of centralised Personal Cloud services, like iCloud and Dropbox, allow limited service usage free of charge, to increase popularity and develop new monetisation mechanisms. Nonetheless, the user service level agreements (SLA) for the class of free users provides minimum guarantees. For example, if a part of the service is compromised and sensitive information are leaked, the SLAs minimize the obligations of the provider towards affected users Declan McCullagh [2011]. Such costs are not directly observed by an end-user, but they can impact significantly his social and work experience.

- *Availability*: Cloud services run on well connected infrastructures, monitored by highly-skilled engineers to ensure performance, functionality and security. Service functionality requires only outgoing Internet connectivity. However, the centralised architecture of such services introduces a single point of failure in inter-device communication. For example, two devices, belonging to different users and connected to the same local network, cannot exchange a file over Dropbox, if the firewall policy forbids connectivity with the Dropbox servers [1].

5.1.2 Design Goals

In the previous section, we present the capabilities and limitations of existing Personal Cloud applications. Decentralized Personal Cloud applications luck user-friendliness and their functionality is subject to the network policy. On the other hand, centralised Personal Cloud architecture introduces privacy vulnerabilities Greenwald and MacAskill [2013] and performance bottleneck.

We believe that an efficient and secure Personal Cloud framework should employ a hybrid approach to scale the problems of connectivity, security and usability. User information should be stored on user-managed devices and the system must provide user-controlled privacy. The wealth of available decentralized Personal Cloud applications provides a sufficient set of applications to support the required functionality. Nonetheless, Personal Cloud functionality cannot ensure Internet-wide inter-device connectivity, without the usage of cloud services. Cloud support is required in order to establish

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

a control channel between the instances of the system. Additionally, data plane end-to-end connectivity can be achieved over the Internet, without cloud mediation, by appropriate testing and configuration of existing connection establishing software. We set the following high-level goals for our system architecture:

- *Connectivity*: A personal cloud architecture must provide connectivity, and thus functionality, in all network environments. The system should be able to recover functionality 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. In addition, the control abstraction should 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 modification of their functionality.
- *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 Personal Cloud-enabling framework with user-controlled security. The system provides an Internet-wide overlay network abstraction between the devices of a user, enabling decentralised Personal Cloud functionality. Signpost uses a cloud-based inter-device control channel to coordinate end-to-end network path establishment between devices using existing connection establishing software and protocol. 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.

Signpost design defines a generic testing and configuration model which automates the process of connection establishment for a wide range of relevant software. The system design provides seamless integration with existing applications. The applications can use Signpost paths by routing traffic to a Signpost specific address, 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 Signpost connection establishment mechanism. Finally, Signpost exposes a security control interface through which users can express a security policy. As a consequence, the user controls the security properties of inter-device information dissemination, while avoiding privacy-sensitive cloud service data offloading.

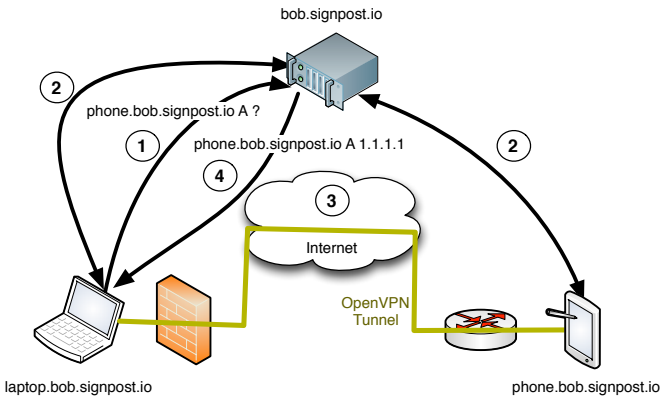


Figure 5.1: A simple example of the Signpost abstraction when the user Alice interconnects a smarthphone with the home computer over the Internet.

In order to introduce the reader to the Signpost abstraction, we depict in Figure 5.1 a simple use case example. In this scenario user Bob, while at work using his smartphone, wishes to access some files on his laptop, situated behind his NATing home router. In order to express his interest to connect to his laptop, Bob performs a name lookup for the domain name `laptop.bob.signpost.io` to his local Signpost resolver (step ①). The name request propagates through the public DNS infrastructure to his Signpost cloud service, which we call the *Signpost controller*. A verified name request to the Signpost controller initiate 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 becomes 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 continues to evaluate different connection establishing mechanism, aiming to optimize the utilized path, and monitors path availability, in order to recover connectivity in case of a disconnection.

5.3 Signpost Architecture

In Figure 5.2 we present a diagram of the Signpost architecture over different viewpoints. In the lower section of the figure, we present the design of the Signpost software and its integration with existing applications. Signpost logic is contained in a single executable, and requires from the guest OS to expose a flow control abstraction, like the OpenFlow protocol, and to redirect DNS queries to the embedded DNS resolver. The software consists of three subsystems: The *Connection Engine* (Subsection 5.3.3), responsible to set up and manage *Network Tactics* (Section 5.3.1 in order to establish end-to-end paths, 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. In the middle layer of Figure 5.2, we present the control plane interconnection between Signpost devices. The system setups an Internet-wide inter-device control channel to enable capability and parameter negotiation between devices. The architecture requires 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 and orchestrate connectivity establishment. Signpost, in addition, provides off-line authenticated connectivity between devices connected to the same network, by exploiting the service discovery functionality of the Bonjour protocol. In the upper layer of Figure 5.2, we present 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. We extend the normal name resolution functionality of the DNS protocol and introduce the *Effectful name resolution* operation for Signpost-enabled domains; a name resolution expresses the

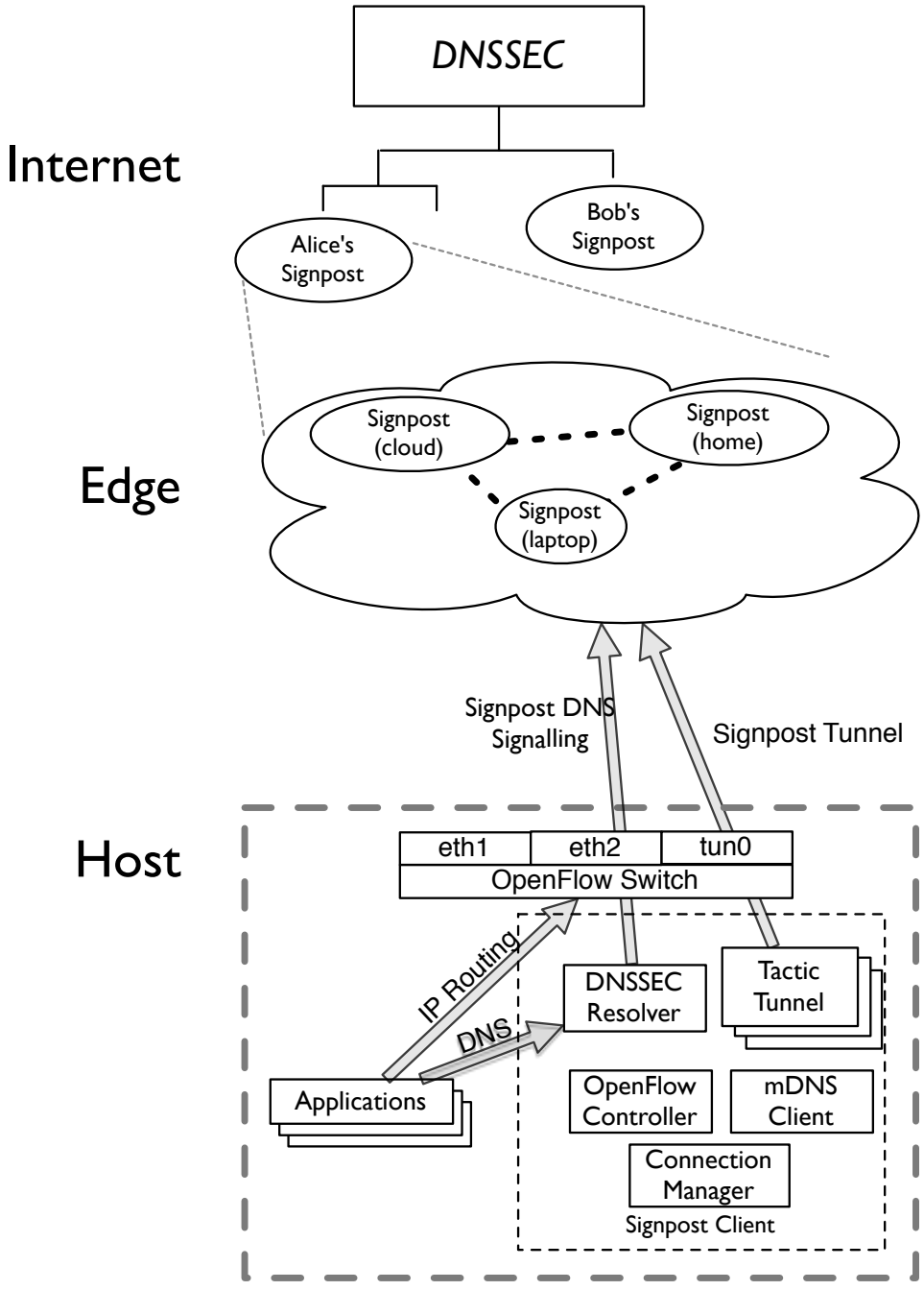


Figure 5.2: Signpost architecture

Tactic name	Purpose	Layer	Transport	Auth.	Encrypted	Anon.	Signpost Support
Avahi	Discover	7	UDP	No	No	No	Yes
Samba	Discover	7	UDP	No	No	No	No
Bonjour	Discover	7	UDP	No	No	No	Yes
Universal PnP	Discover	7	UDP	No	No	No	Yes
dns2tcp	Tunnel	7	UDP	No	No	No	No
DNScat	Tunnel	7	UDP	Yes	No	No	No
HTTP-Tunnel	Tunnel	7	TCP	No	No	No	No
iodine	Tunnel	7	UDP	Yes	No	No	Yes
NSTX	Tunnel	7	UDP	No?	No	No	No
Proxytunnel	Tunnel	7	TCP	Can be	Can be	No	No
ptunnel	Tunnel	4	ICMP	Yes	No	No	No
tuns	Tunnel	7	UDP		No	No	No
SSH	Tunnel/Encrypt	7	TCP	Yes	Yes	No	Yes
IPSec	Tunnel/Encrypt	3 (4*)	IP	Yes	Yes	No	No
OpenVPN	Tunnel/Encrypt	7	UDP/TCP	Yes	Yes	No	Yes
libjingle	Nat punch	7	UDP/TCP	Yes	?	No	No
privoxy	Anonymize	7	TCP	?	?	Yes	Yes
tor	Anonymize	7	TCP	No	Yes	Yes	Yes
stunnel	Encrypt	7	TCP	Yes	Yes	No	No
TCPCrypt	Encrypt	4	TCP	No	Yes	No	No

Table 5.1: Tactics table.

interest of a user to establish an end-to-end path (Subsection 5.3.4). For the rest of the section we present in detail the architecture of the Signpost system.

5.3.1 Network Tactic

In order to provide end-to-end connectivity, Signpost reuses the wide range of free and open source connection establishing mechanisms developed by the research community. In Table 5.1, we present a small survey of such mechanisms along with their network requirements and security properties. The table entries exemplify the high diversity between available connection mechanisms. For example, a significant subset of the mechanisms ensures connectivity over strict network policies. An equally significant subset of the mechanisms enhances end-to-end Internet path security and privacy, while a third class of such mechanisms enables connectivity through service advertisement. Further, the mechanisms vary significantly on the operating network layer and the exposed connectivity abstraction. Connection mechanisms expose connectivity over a specific transport layer port or through a network layer device. The majority

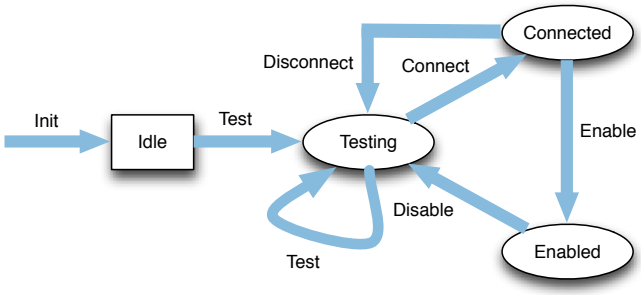


Figure 5.3: Signpost tactic lifecycle

of the mechanisms uses UDP or TCP sockets, but there are mechanisms using other network protocols, like ICMP. Finally, user authentication is achieved through various mechanism. Approaches vary from user-based authentication using either passwords or certificates, to simple Pre-shared passphrases, while a subset of the mechanisms lucks authentication support.

In Signpost we have developed a generic model to encapsulate the testing and control requirements for connection establishing mechanisms, which we call *Network Tactic*. Using this model, we aim to automate the setup process for end-to-end paths. A Signpost Network Tactic is modelled as a 4 state automaton. The state space diagram is presented in Figure 5.3. A Tactic is initialized in the Idle state. A test method invocation transfers the tactic to the testing state and runs the tactic-specific connectivity tests to discover network policy and configuration requirements. If testing is successful, the Tactic is able to transfer to the connect state through an invocation of the connect method. The connect method is responsible to configure an end-to-end path using the tactic connection mechanism. Once the end-to-end path is established, the Tactic can progress to the enabled state and forward traffic over the path. Finally, the Tactic automaton provides methods to backtrack from each state, tearing down establihed paths and clearing stored state.

Furthermore, because we require a mechanisms to allow distributed synchronisation and execution of tactics, we, additionally, define an architecture to split tactic functionality between the Signpost controller and client. The functionality of a Signpost tactic is split logically in two layers: the Southbound layer, implementing low

level tactic operations, and the Northbound layer, translating the tactic abstraction into low level operations. The Northbound layer runs on the Signpost controller, while the Southbound functionality runs on the Signpost clients. The two layers of the tactic, communicate over the control channel using a Json-RPC-based tactic-specific protocol.

As an example of this functionality split, we present the Test method of the OpenVPN Tactic, an IP tunnelling mechanism using certificate-based authentication and connectivity over TCP or UDP. In the test section of the Southbound layer, the tactic implements two methods: the *init* method, which initializes an OpenVPN server with default configuration, and the *t*est method, which tests client accessibility to an OpenVPN server on a specific IP and port. The Northbound layer of the tactic expose a single test method, which uses the Southbound methods to test OpenVPN connectivity towards the other device. The first client returning a successful test result becomes the client of the OpenVPN tunnel. If the test times-out for both devices, then the controller initialises an OpenVPN server and instructs the devices to test OpenVPN connectivity towards the controller.

5.3.2 Forwarding

Signpost connectivity is exposed to applications at the network layer of the host, providing strong backward compatibility with existing decentralized applications. A Signpost cloud is abstracted as a local subnet and persistent local IPs are allocated to each user device. Signpost has a fundamental requirement on each device, for a network forwarding control mechanism, like the OpenFlow protocol, in order to exercise dynamic network control on each host. The Signpost network control is responsible to regulate traffic forwarding and detect connectivity degradation.

The core of the Signpost system exercises minimal control over network traffic and primarily functions as a multiplexer of control traffic between Tactics. Non-Signpost traffic is routed normally by the network stack. For Signpost traffic, the Signpost agent delegates network control to the Tactic which is responsible to provide connectivity towards a specific remote Signpost host. Tactics can exercise control either pro-actively or re-actively and, additionally, they are able to inject and intercept traffic. In section 5.4.1, we present in detail integration examples between the Signpost core design

and various connection establishing mechanisms and discuss how network integration is achieved through appropriate control plane design. Finally, in order to reduce broadcast traffic on the Signpost overlay, we integrate in the control plane of the host a simple ARP proxy which replies with the MAC address *fe:ff:ff:ff:ff:ff* to ARP request for Signpost IP addresses.

5.3.3 Connection Manager

End-to-end path management in Signpost is encapsulated in the Connection Manager Subsystem. The Connection Manager controls path establishment, selects the optimal path between a pair of devices and ensures connectivity. 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 used to form the path. Tactic weight, in the current implementation of the system, are empirically defined through a set of simple rules of thumb. Tactics using the Controller as a relay in the forwarding path have a higher weight than direct paths. Similarly, Tactics using connectionless protocols, like UDP, are considered more efficient than Tactics using connection-oriented protocols, like TCP. Using these simple rules of thumbs we are able to select the most efficient path, but more accurate performance characterisation mechanisms can be developed using active network measurements.

Additionally, Signpost defines a simple security model for users to express security requirements. Security in Signpost has three dimensions: *encryption*, *authentication* and *anonymity*. Each Tactic provides a subset of these three properties, based on the capabilities provided by the underline connection mechanism. An encrypted Tactic applies strong cryptography on both directions of the end-to-end path. An authenticated Tactic uses strong authentication to establish end-to-end paths. Finally, an anonymizing Tactic provides end-to-end paths which obfuscate user identity from eavesdroppers. Additionally, in order to enable security flexibility, the Signpost architecture allows *Tactic synthesis*. An end-to-end path can be constructed through appropriate layering of multiple connectivity establishing software. For example, an anonymized and encrypted path can be established through an SSH tunnel running over a TOR cir-

cuit between the two devices. 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.

Signpost is able to establish multiple end-to-end paths between two devices. Unfortunately, multipath connectivity is not supported by popular transport protocols and newer protocols like SCTP and multipath TCP, supporting multipath connectivity, are not yet available in production systems. As a result, in Signpost we establish and use a single end-to-end path between any pair of devices. Signpost path search runs on the Signpost-controller and uses a simple width-first search algorithm over the complete space of Tactic combinations. The search is initiated by the connect method of the Connection Manager module, with parameters the names of the devices and a list of security properties. The Manager is responsible to return an initial end-to-end path and continuously try to improve it.

The logic of the search algorithm is pretty simple. During a connection request the system spawns a thread for each Tactic, testing end-to-end connectivity. If the test is successful, the Tactic is instructed to connect the two end-points. If the connection is successful and either there is no other Tactic enabled or the currently enabled Tactic has a higher performance weight, 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 recursively tests tactic connectivity over the established path to find Tactic synthesis with sufficient security properties. The Manager returns a successful result, when the search establishes an end-to-end path compliant with the security policy. In addition, the Manager will continue to search for other lower weight paths between the two devices. The recursion terminates if all remaining Tactic combinations have a higher performance weight or they are synthesized by more that three Tactics. As a result, the Manager can establish rapidly an end-to-end path between two devices and progressively optimize path performance, while the devices remain interconnected.

Finally, in the connection manager we integrate a connectivity monitoring thread. The thread is responsible to detect if a Tactic is not functioning properly and report to the connection manager. We employ a simple mechanism to detect malfunctioning tunnels. Initially, the monitoring thread use the `flow_stats` OpenFlow message to check if any data have been exchanged during a monitoring period and thus infer that

the path is working properly. If the path remained idle during the monitoring period, the thread will try to actively test connectivity through a UDP probe towards a UDP heartbeat service running on each Signpost daemon. If the heartbeat times out, then the connection manager will teardown the path and resume the path establishment process. In addition, a Tactic can also implement application specific connectivity testing, e.g. OpenVPN provides a mechanism to report the state of a tunnel.

5.3.4 Effectful Naming

The majority of Internet-connected devices are essentially anonymous from a network perspective and 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 is slow and, thus, limited. A fundamental requirement for the Signpost architecture is to support stable secure resolution of device names into concrete end-to-end paths. Signpost naming functionality is established through the DNS protocol Mockapetris [1987] for two main reasons. Firstly, DNS is an effective solution for the naming problem. DNS is an Internet-wide and accessible service, which is never blocked by the local network. Additionally, the introduction of the DNSSEC protocol extensions provides a sufficient mechanism for bi-directional authentication. On the other hand, the DNS protocol carries significant information on the user connectivity intentions. The Internet naming service is a delay-tolerant mechanism reflecting the interest of applications to connect to a host. DNS domain names represent fundamentally Internet services, decoupled from the network layer technology, while the naming format provide a good match to spoken language.

Domain naming 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 (RR) and provide a wide range of information for the specific domain name like its network addresses, possible name aliases and lists with available services. In order to scale the performance of the naming service across the Internet, the design of the system allows delegation of service for a specific subtree of the naming hierarchy to other servers through the NS RR type. In addition, the DNS service defines a record caching mechanism, to improve lookup

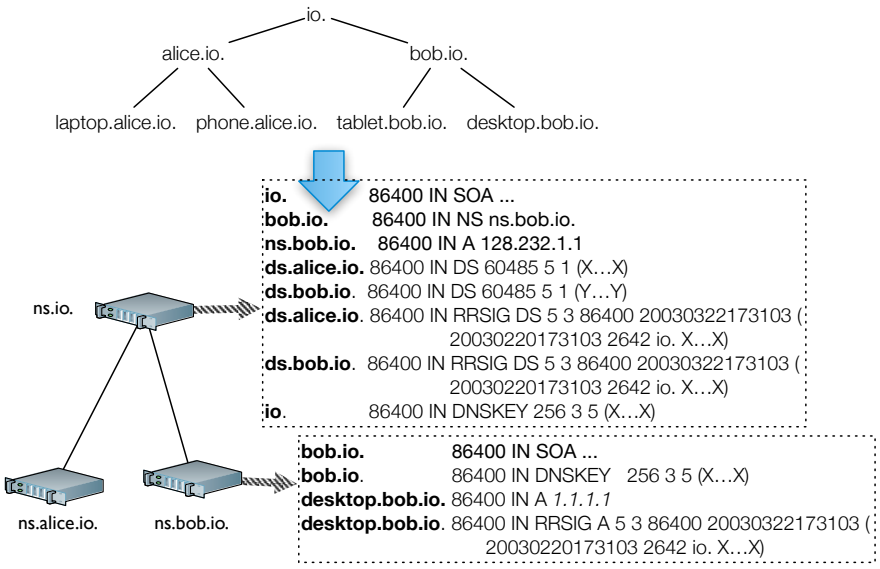


Figure 5.4: DNSSEC key configuration for the io. and bob.io. domains. The io. nameserver zone file contains a DS RR with the hash of the bob.io. DNSKEY RR and an RRSIG RR signature of the DS RR signed with the DNSKEY of the io. domain. bob.io. uses its DNSKEY to sign an A RR for desktop.bob.io.

performance and service load.

Secure Names For All Internet Users In the initial definition of the Internet naming service, the protocol provided very weak security guarantees. In order to enhance the security primitives the IETF standardised a number of DNS protocol extensions, described as DNSSEC. DNSSEC defines three RR types Arends et al. [2005], enabling authenticated response for the protocol. We present the authentication mechanism in Figure 5.4. In DNSSEC, each zone owns at least a cryptographic key, used to sign authoritative data through public key cryptography. RRSIG RR types are used to carry RR signatures, while the NSEC RR type can be used to inform a resolver for the luck of a respective record type. The DNSKEY RR type disseminates the public keys of a server to resolvers to authenticate record signatures and the DS RR type allows a domain to express his trust to the signing keys of another domain. With respect to Figure 5.4, the domain `bob.signpost.io` signs an A record for the host `desktop.bob.signpost.io` through an RRSIG RR using bobs DNSKEY. The bobs key is registered in the io. domain with a DS record containing the hash of bob's

key and signed with the signing key of the io. domain. As as result, the DNSSEC extensions form a chain of trust within the naming service, ensuring the authenticity of any query response . The resolver requires, as in the X509 certificate architecture, only a list of authenticated anchors configured out-of-band of the protocol and injected in the authentication chain.

Signpost uses extensively the DNSSEC RR types to establish an authenticated control channel between the devices of the user. For each user of the Signpost system, we require control delegation of a domain to the Signpost controller and registration of the domain 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 domain `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-Merkle-Hellman Rescorla [1999] and derive new shared private keys between any two devices over unsecure channels.

The base DNSSEC RR types provide a mechanism to authenticate responses from a nameserver to a DNS resolvers. In terms of the Signpost architecture, we are interested additionally to authenticate 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 [2000], to sign DNS requests using the key of the device.

Fitting DNSSEC in Signpost Signpost evolves the default DNS processing logic on the client and the controller of the architecture. On the controller we develop a programmable DNS server functioning as the authoritative server for the user domain. The server, beyond serving domain records and signing responses on-the-fly, can additionaly verify SIG(0) signed queries and translate 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 `desktop.alice.signpost.io`, will

[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]*

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 triger a recursive search of the DNS name tree, in order to retrieve required RRs. For Signpost host requests, the resolver sings the query with a SIG(0) record using the private key of the device.

Using the DNS protocol we are also able to provide offline functionality for two devices connected to the same subnet but disconnected from the Signpost Controller. In order to implement this, we use DNS-based service discovery (DNS-SD) Cheshire and Krochmal [2011b]. DNS-SD is an RR organisation specification which enables service advertisement and browsing over the Multicast-DNS Cheshire and Krochmal [2011a] service and effectively providing an efficient local service discovery mechanism. The combination of DNS-SD and multicast-DNS is currently a popular service discovery mechanism in local networks, supported by most operating systems. The Signpost system 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 domain. Another Signpost client can verify a destination Signpost service, using his locally cached controller public key, and establish a control channel. During the offline mode, the query receiver will function as a Signpost Controller, responsible to server RR request for its domain name only.

5.3.5 Security and Key Management

Signpost develops a public key distribution mechanism using the DNSSEC protocol and, thus, enables seamless inter-device authentication and security bootstrapping. We use a three layer key hierarchy to enable dynamic trust control and device key revocation. Signpost key hierarchy extends the key hierarchy defined in the DNSSEC protocol. A Signpost cloud has at least two RR signing keys: A *Zone Signing Key (ZSK)* and a *Device Signing Key(DSK)*. ZSK is used to sign the DSK, while the hash of the ZSK is stored as a DS RR in the top-level name servers. DSK is used by the controller to sign device keys, as well as, authoritative RR. The separation of the authentication mechanism between two keys allows a user to maintain a persistent anchor in the DNSSEC key chain and, in parallel, be able to revoke trust for a key of the domain without having to propagate the change to a server beyond the user control. Each

registered Signpost device must construct a Device Key and add it securely in the zone file of the Controller, to join the user cloud. The device public key is added in the zone file of the Controller through a DNSKEY record and the controller will generate an RRSIG RR to sign the key and, effectively, allow other devices to verify the identity of a device. 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 analyse the performance of the Signpost system and present our experience from using a number of distributed Personal Cloud applications, replacing popular centralised Personal Clouds. Specifically, we present the details of a strawman implementation of Signpost (Section 5.4.1), measure the performance of the Signpost Tactics over the Internet (Section 5.4.2) and present a small scale study of the functionality of current application over the Signpost system.

5.4.1 Signpost implementation

Signpost is implemented predominantly in Ocaml and uses existing protocol libraries to integrate DNS and OpenFlow protocol support to the core daemon of the system. More specifically, we employ the ocaml-dns library for DNS server and client functionality, the cyptokit library for cryptographic key manipulation and the ocaml-openflow library for OpenFlow controller implementation. Signpost also uses a portion of C code to implement binding with the OS routing stack. Our strawman implementation is fully functional under Linux and Android, using the Open vSwitch switch implementation, and MacOSX, using the userspace switch implementation provided by the ocaml-openflow library.

The Signpost Tactic model is able to encapsulate the testing and configuration requirements for the connection establishing mechanisms:

- *Direct*: The Direct Tactic implements method to evaluate the ability of two device to connect directly over the network without any tunnelling mechanism. After a name lookup, the tactic tests if direct bi-directional connectivity is possible between the two devices for a set of popular ports (e.g. TCP and UDP

connectivity through the ports 53, 80, 443 and 8080). If the tests are successful, the tactic insert a series of OpenFlow rule on both devices, which translate Signpost device addresses to network accessible addresses.

- *OpenVPN*: The OpenVPN tactic establishes connectivity using the OpenVPN tunnelling mechanism. During testing, the tactic tests connectivity over UDP to port 1194 between the devices, as well as, towards the controller. If the test is successful, Signpost devices use the Signpost key infrastructure to generate public keys and bootstrap the OpenVPN authentication mechanism. Because of some limitation on the certificate chain evaluation of OpenSSL, during a connection the Tactic generates transient private keys, signed by the user private key, while the tactic injects in the trusted certificate configuration of the OpenVPN software a key certificate of the destination device signed by its private key. The tactic configure the OpenVPN software to expose connectivity over an Ethernet TAP device Krasnyansky et al. [2002], which is added under the control of the OpenFlow switch. Because the OpenVPN software contains a small ARP cache, we assign a unique IP in the 10.0.0.0/8 subnet to each TAP device and broadcast appropriate ARP notification packets to create appropriate state in the cache. The tactic, similarly as in the Direct tactic, inserts OpenFlow rules on all participating devices to forward packet over the OpenVPN tunnel, while translating appropriately the source and destination addresses.

- *SSH*: The tactic provides inter-device connectivity over the SSH protocol. For this tactic we configure and run an SSH daemon on every Signpost device on port 10000, permitting only key-based tunnelling connectivity. Tactic testing evaluates the ability to establish a TCP connection between the device or through the Controller. Connectivity enforce strong user authentication by using the key-based authentication of the SSH protocol. Signpost device servers append device public keys in the authorized_key file in an ad-hoc manner and clients use the private key of the device upon connection on their SSH client. The SSH tactic associates the SSH tunnel with a local TAP Ethernet interface on each device and the tactic controller insert appropriate OpenFlow rules to forward traffic towards the relevant TAP interface.

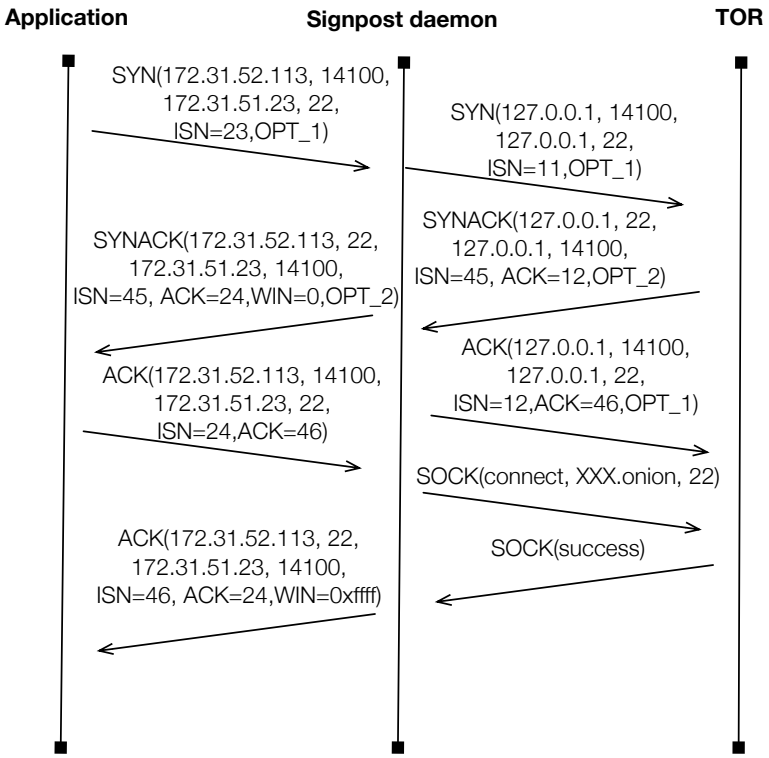


Figure 5.5: TOR tactic implementation in Signpost. The local Signpost daemon translates new TCP connection requests towards the destination device to appropriate SOCK request to the SOCK proxy of the software.

- *Privoxy*: This tactic enables HTTP request anonymization using the Privoxy Privoxy [2013] HTTP proxy. The tactic configures a privoxy HTTP proxy server on each device, configured with a strict policy to anonymize privacy leaking HTTP headers. In order to establish connectivity, the tactic handles TCP flows in a proactive manner. For each SYN packet, the tactic install bi-directional flows, forwarding data from the application to the local listening port of the Privoxy daemon.

- *TOR*: The TOR tactic enables connectivity between devices over the Anonymized network of Tor Dingledine and Mathewson [2006]. The tactic configures a TOR client on each device, providing a SOCKS proxy which forwards TCP traffic through the TOR network. Additionally, the tactic takes advantage of the hidden service functionality of the TOR system, which allow TOR nodes to run anonymized services. In order to achieve this, TOR generates for each host an

anonymized domain name under the .onion domain and employs the embedded name lookup capability of the SOCKS protocol. The TOR testing method is primarily responsible to initialize the TOR daemon, propagate the domain name of the device to the Controller and test the effectiveness of the tactic in the network environment. Testing and setting up of TCP flows using the TOR tactic follows a simple OpenFlow-based mechanism. The TOR initially insert an OpenFlow rule to forward all packets towards the Signpost IP of the remote host to the controller. Upon a SYN packet transmission from an application to the remote Signpost host, the tactic will initiate a TCP connection with the local SOCKS proxy with a request to establish a path between the two devices. In parallel, once the TCP connection is established with the local SOCKS proxy, the tactic will respond to the initial SYN request with a SYNACK packet with appropriately modified sequence and ACK numbers to accommodate the additional bytes of the SOCK protocol interactions with the proxy. Additionally the packet carries zero sized window, in order to suppress any further data transmissions from the application. Once a SOCKS response is returned by the TOR proxy, the tactic will either respond to the initial TCP flow with an ACK with a non-zero window and insert appropriate OpenFlow rules to permit connectivity between the two device over the SOCKS tunnel, or the Signpost daemon will inject a TCP RST if the SOCKS response was unsuccessful. A graphical representation of the TOR tactic is presented in Figure 5.5. In order to replicate a similar abstraction for UDP and ICMP traffic, we additionally setup a VTUN tunnel over a TCP connection, thus enabling IP-level connectivity. Nonetheless, we avoid using the tunnel for TCP connections, to reduce any capacity loss due to the additional headers of the tunneling mechanism.

- *DNS-SD*: The DNS-SD tactic enables service advertisement between Signpost devices. DNS-SD Cheshire and Krochmal [2011b] is a common OS service enabling hosts to advertise services in the local network. The main motivation for this tactic is to aid existing decentralised Personal Cloud applications, which depend on the DNS-SD functionality, to discover the service capabilities of connected Signpost devices. The Tactic connect method primarily intercept DNS-SD packets from the local stack of the device and propagates them over

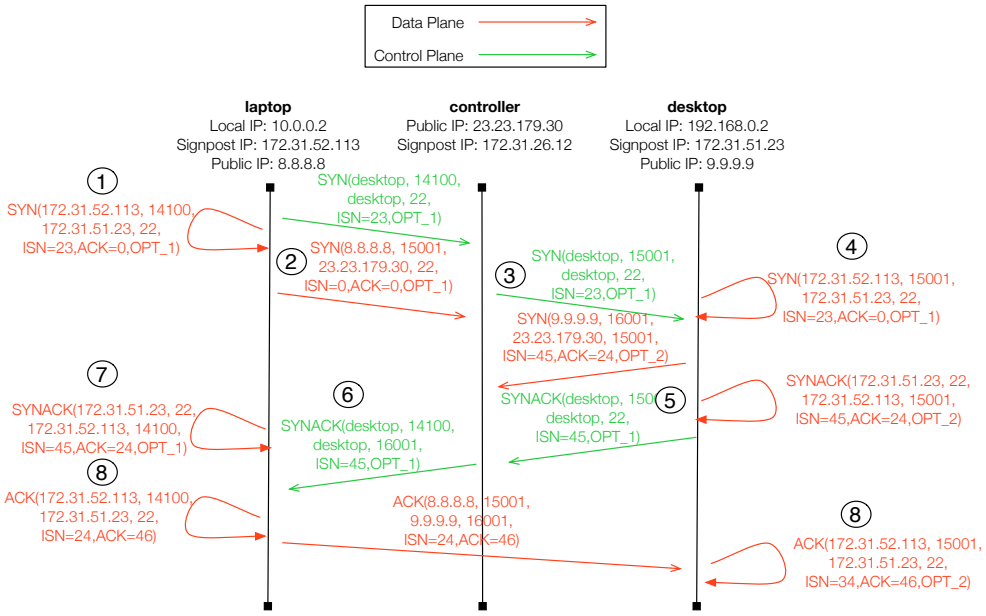


Figure 5.6: NAT punch Tactic implementation in Signpost. The device local Signpost daemon injects crafted packets to create appropriate state in the local network NAT, while the controller enables the Tactic to detect the incurred NAT port mapping for both hosts of the path.

the control channel to the other devices of the user. Signpost device must inject equivalent DNS-SD multicast packets through the OpenFlow protocol to the network loopback device and effectively to the local DNS-SD daemon.

- *NAT-punch*: This tactic implements a packet injection technique which allows TCP and UDP flows to bypass NAT boxes. The functionality of this tactic is limited to Full-cone NAT, Address-restricted NAT and Port-restricted NAT. The NAT-punching tactic uses the Controller to infer the port mappings configuration of the NATbox. We present a schematic of the mechanism for a TCP connection in Figure 5.6. Specifically, for new TCP connections, the daemon intercepts the SYN packet (Step ①), and propagates the important TCP header parameters (Port number, initial sequence number and TCP options) over the control channel to the remote device and in parallel send a SYN packet from the device to the Controller in order to infer the port mapping applied by the network NAT and create appropriate state in the NAT (Step ②). The controller will extract the

Tactic name	Direct connectivity			cloud-based connectivity		
	setup (sec)	throughput (kbps)	RTT (msec)	setup (sec)	throughput (kbps)	RTT (msec)
Direct	1	94.1	1.5	-	-	-
OpenVPN	13.3	66.0	1.23	20	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 evaluation in terms of latency to setup a path, throughput and RTT time.

source port mapping applied by the NAT and forward it, along with the TCP initial sequence number, to the destination device, upon the receipt of the SYN packet from the flow initiating device (Step ③). Once the destination device receives the control message from the controller, it will generate a SYN packet, using the header fields of the initial SYN packet, and send it 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 propagate 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 propagate to the initiating device (Step ⑥), which will inject a SYNACK packet to the local stack to progress the TCP-handshake (Step ⑦). Finally, the two device insert appropriate entries in the OpenFlow flow table to translate 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 over the control channel to the remote device, as well as, transmit an similar packet to the Controller in order to generate appropriate state in the intermediate NATboxes. The remote device will also send a similar UDP packet to the Controller Once the controller has received both UDP packets, it will notify both devices to insert appropriate flow rules to transform IP address and port numbers of the flow and resume UDP traffic transmission.

5.4.2 Tunnel Evaluation

In order to evaluate the performance of our strawman implementation, we conduct a series of measurements to quantify the impact of the architecture to network func-

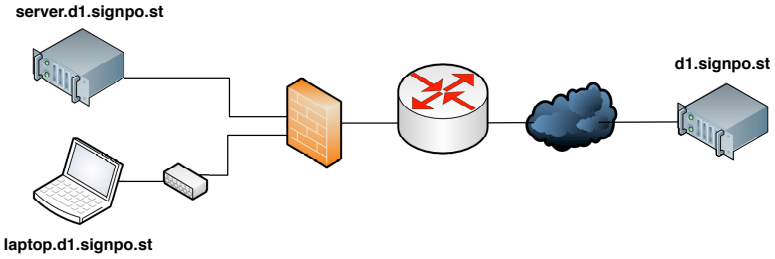


Figure 5.7: Signpost tunnel evaluation topology. *server* and *laptop* hosts are connect in the same network but on different subnets with a firewall enforcing a network security between the two devices. The Signpost *controller* is hosted on an Amazon ec2 micro instance.

tionality, as well as, the performance of each Tactic. Our testbed consists of three hosts, two clients and a cloud controller. The two Signpost clients (laptop.d1.signpo.st, server.d1.signpo.st) are connected to the production network of the Computer Laboratory on different subnets of the network. The network enforces a security policy both between the devices,as well as, towards the Internet through a stateful firewall. Additionally, one of the hosts is connected to the network through a NATed home router. The Signpost controller of our experiment runs on an Amazon EC2 micro instance. The topology of the measurement is depicted in Figre 5.7. Using Signpost, we establish all possible end-to-end tactics [1] and measure a series of important network performance parameters. 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 latency to setup the path after a name resolution from a host, the throughput of the path, using an Iperf TCP measurement probe, and the average RTT of the path, using an Iperf UDP measure probe of 1 Mbps. We contact our measurement ten times for each possible configuration and present the median value of the measurements in Table 5.4.2.

From the result of our experiment we note the difference in capacity and latency when a path is setted up through the Signpost controller. Edge network connectivity provides two to three orders of magnitude higher performance than cloud-aided paths. In addition, we highlight the trade-off between security and performance between Tac-

[1]We exclude the privoxy and DNS-SD tactics, as they don't provide end-to-end connectivity, but focus more on providing additional functionality to a path

tics. For example, although the TOR Tactic provides strong anonymization functionality, its performance and latency are inadequate for a significant portion of modern network applications. The performance degradation is primarily due to the onion routing scheme used to obfuscate the destination of a network packet, as well as, due to the sharing nature of the system. Nonetheless, the performance and security trade-offs are exposed and controlled by the end-user through the Signpost control abstraction. Furthermore, based on the measurement of the path setup latency, we conclude that the Signpost control plane logic has minimal impact on the host network functionality. Bootstraping an end-to-end path through Signpost incurs noteworthy setup latencies for new device paths but the DNS naming service is designed to handle such latencies [1]. Only the TOR tactic incurs significant latency to setup an end-to-end circuit over the onion overlay. At the moment our implementation exposes this latency to the user application, in order to avoid stale TCP connections. This latency can be easily hidden by the application if a more optimistic flow setup approach is implemented (e.g. Assume that an end-to-end connection will be setup between the two devices and block application by advertising early a zero length window through the SYNACK packet). Finally, using the aforementioned topology we also measured the performance of Tactics synthesis. From our experiment we 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 order to evaluate the effectiveness of the Signpost system, we tested our strawman implementation compatibility with a number of popular applications providing resource and information sharing.

File Sharing In order to evaluate the functionality of file sharing applications over Signpost we tested three popular data sharing applications: NFS Shepler et al. [2010], git-annex git [2013] over rsync and file transfers over the SSH protocol. We report that

[1] Linux has a default timeout of 5 seconds and 2 retries before timing out a name resolution, providing a budget of 10 seconds to a Signpost Tactic to provide a response http://linux.die.net/include/resolv.h

all mechanisms function as expected, with noticeable performance differences proportional to the throughput provided by each Tactic. In our initial experimentation we had to modify mildly our flow management logic, as the SSH protocol tends to modify the IP header ToS bits during the transmission of a flow and requires a wildcarded flow definition.

Resource sharing Using the topology in Figure 5.7, we setup and test the support of the system for resource sharing applications. More specifically, we test support for CUPS remote printing, DAAP media sharing using the UPnP resource advertisement, VNC remote desktop and SSH remote access and conclude that our Signpost architecture support the application functionality.Nonetheless, some of the Tactics incur significant RTT delays which impacts significantly the user experience. For example, the SSH software exhibited significant responsiveness problem when routed over a TOR path.

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 between the increasing number of personal devices, a generic abstraction which we term Personal Cloud. We elaborate on the available mechanisms providing such functionality and highlight the trade-offs among the available solutions. We argue that an evolve control plane for end-user devices can provide a sufficient and secure framework to enable Personal Cloud functionality, and present Signpost, an evolved control plane for end-user devices. Signpost provides network-level secure and user-managed device interconnection by automating the configuration task of available network connectivity mechanisms, providing sufficient, secure and backwards-compatible network functionality to existing distributed Personal Cloud applications.

Signpost evolves the control plane of the user devices on two aspects. Firstly, the device control plane is integrated with *Network Tactics*; mechanisms enabling ad-hoc end-to-end connectivity, like tunneling software and NAT punching techniques. The architectures develops a generic model coupled with a secure Internet-wide control channel which provides the ability to orchestrate the testing and configuration of Tac-

tics between devices. In addition, the rich security properties provided by existing network Tactics, allow the system to expose a security control abstraction, enabling users to define the trade-off between performance and security when they interconnect their devices. Secondly, the architecture integrates the control plane logic with the network naming service. As a result, Signpost provides Internet-wide persistent device names, while DNSSEC naming extensions allow Signpost to construct a secure and globally accessible key distribution mechanism.

We evaluate our strawman Signpost implementation in terms of its ability to integrate existing network connectivity mechanisms, its impact in user perceived performance and its ability to replicate existing Personal Cloud use cases. We provide evidence on the generality of the Signpost Tactic model by presenting its integration with a number of connectivity establishing mechanisms, namely OpenVPN, TOR, SSH, NAT punching and DNS-SD. Additionally, we measure the performance of integrated Tactics in a typical deployment scenario and verify the backward compatibility of the system with a number of popular decentralised applications. In the next chapter, we conclude the results of our research and discuss future direction of the research.

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 control plane performance scalability using the Software Design Networking paradigm. Chapter 1 began by motivating the significant bottleneck of current network technologies on the control plane. We argued that control plane performance has multiple aspects (e.g. responsiveness, usability, security), which have variable importance and depend highly on the deployment environment. It was concluded that an effective control plane architecture must take under consideration such requirements, as well as, the properties of the environment during the design of efficient control plane architectures.

Chapter 2 then considered background and related work to the problem of network control performance scalability. We provided a bottom-up design discussion on available network control mechanisms. We, initially, elaborated on the architecture of current network devices and presented a generic design model of the integration between the control and data plane in a single devices. 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 and user friendly control is reduced, especially as data-plane rates increase exponentially. Furthermore, we presented a series of experimental approaches which combat a series of network

control limitation and provide flexible, distributed and evolvable control. Namely we present Active Network, Devolved Control of ATM Network 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 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 SDN experimenters to evaluate the performance of SDN architectures. More specifically, we presented OFLOPS, a high-precision hardware-accelerated OpenFlow switch measurement platforms 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 develop and run a series of tests characterising the elementary functionalities of the OpenFlow protocol and detect significant variance between switch protocol implementations which can affect significantly the performance of an OpenFlow-based control architecture. Motivated by the variability between the implementations of the OpenFlow protocol, we have developed the SDNSIM platform. The SDNSIM platform provides a high precision experimentation environment, which allows users to implement their control logic and traffic models and evaluate the performance of the system using specific switch performance profiles. Using SDNSIM, we replicate the functionality of a small-size datacenter networks and evaluate the effectiveness of distributed hierarchical control-plane architectures, finding minimal impact in the data-plane performance.

Chapter 4 elaborated on the problem of management scalability in modern home networks. More specifically, using ethnographic and measurement studies of the home setting, we identified a significant mismatch between the user requirement and understanding of network functionality and the existing technologies. Motivated by this observations, we redesign the home router, implementing a series of protocol modifications which enhance user control and bridge the user network perception with the underlying functionality. Additionally, we extended our proposed architecture and presented a distributed resource control mechanism which integrates users application-level requirements with the ISP policy, in an effort to develop a user-friendly control scheme for the last-mile bottleneck in current residential broadband networks. We pre-

sented the development of a strawman implementation of our system and verified that the architectures incurs minimal impact on network functionality, while the protocol modifications remain backwards compatible with a number of popular devices, OSes and applications. In addition, our QoS mechanism was able to provide better support for latency sensitive application in the presence of bursty traffic.

Chapter 5 discussed Internet-scale connectivity scalability. Through the work of this chapter, we aimed to develop a decentralised and Internet-wide Personal Cloud between the devices of users. In this chapter we described the limitations of existing Cloud platforms 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, an evolved control plane for end-user devices providing secure, continuous and decentralised inter-device connectivity. Signpost enables local device controllers to establish a global control channel, test and negotiate connectivity capabilities between devices and establish ad-hoc secure end-to-end data plane paths between devices using off-the-self connection establishing mechanisms. Furthermore, we presented an implementation effort of the Signpost architecture, enabling support for OpenVPN, TOR, SSH, Privoxy, NAT punching and DNS-SD functionality. Using the Signpost strawman implementation, we verified the low impact of the architecture to network functionality and its backwards compatibility with existing applications.

6.2 Future Work

This section notes areas where further work is required, and future directions related work could take. The first such area, and one which applies on the results of Chapter 3 is the definition of a distributed scalable network control framework. Hierarchical control, as presented through the evaluation experiment using the SDNSIM, has low impact on network performance, but a complete architectural design requires further exploration of mechanisms providing strong guarantees on control plane responsiveness. A series of approaches have been proposed in the field, e.g. nicira edge-based control architecture Koponen [2012], provide a performable approach to reduce core network control load, but we believe that a more dynamic mechanism is required by

network operators in order to control more the trade-offs between traffic control and performance. Nonetheless, we believe that the SDNSIM platforms provides a sufficient framework to evaluate the impact of such control distribution mechanisms.

Additionally, a number of areas for further work arise from the work in Chapter 4. A first research effort is to evaluate the scalability of the control scheme. The control logic is capable to handle multiple switches, but there is still reasonable complexity to integrate other network control protocol with the router design, in order to introduce our control architecture in larger network settings. In addition, the novel control architecture of our router provide novel opportunities to augment home network functionality . Network design can be extended to enable seamless mobility to home network users when connecting to remote wifi networks and the user can carry his network configurations to the new network. On the other hand, interesting opportunities arise if we extend the homework QoS model throughout the ISP network. In parallel, a interested economical model can be developed to reuse network resources which remain unutilized by the homeowner for a period of time, e.g. exchange high priority traffic for specific timeslots with neighbours.

Finally, a number of pieces of work arise from Chapter 5. The Signpost design at the moment is limited in its policy expressiveness, which though 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 the Signpost architecture can be used to reflect higher level social relationship and increase the in-network flexibility. For example, the devices of a user who is a member of the University of Cambridge can connect through the Signpost architecture with the university network control plane, when a device is connected to the university network. At run-time the device Signpost daemon can negotiate with the university Signpost controller the device network requirements and reflect them in the University network control.

References

git-annex. http://git-annex.branchable.com, 2013. 130

B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, H. Levkowetz, and Ed. Extensible Authentication Protocol (EAP). RFC 3748, IETF, June 2004. 89

Aditya Akella, Srinivasan Seshan, and Anees 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, New York, NY, USA, 2003. ACM. ISBN 1-58113-773-7. doi: 10.1145/948205.948219. URL http://doi.acm.org/10.1145/948205.948219. 96

Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, SIGCOMM '08, pages 63–74, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-175-0. doi: 10.1145/1402958.1402967. URL http://doi.acm.org/10.1145/1402958.1402967. 70

Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin 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, Berkeley, CA, USA, 2010. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1855711.1855730. 38

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. ISSN

0890-8044. doi: 10.1109/65.690959. URL http://dx.doi.org/10.1109/65.690959. 23

D Scott Alexander, Bob Braden, Carl A Gunter, Alden W Jackson, Angelos D Keromytis, Gary J Minden, and David Wetherall. Active Network Encapsulation Protocol (ANEP). Experimental, 1997a. 22

D. Scott Alexander, Marianne Shaw, Scott M. Nettles, and Jonathan 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, New York, NY, USA, 1997b. ACM. ISBN 0-89791-905-X. doi: 10.1145/263105.263149. URL http://doi.acm.org/10.1145/263105.263149. 23

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. ISSN 0163-6804. doi: 10.1109/35.722141. 23

L Andrerson, I. Minei, and B. Thomas. RFC 5036: LDP Specification. Submitted, 2007. 19

Arvind Arasu, Shivnath Babu, and Jennifer Widom. The CQL continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15(2): 121–142, June 2006. ISSN 1066-8888. doi: 10.1007/s00778-004-0147-z. URL http://dx.doi.org/10.1007/s00778-004-0147-z. 82

Roy Arends, Rob Austein, Matt Larson, Dan Massey, and Scott Rose. RFC 4034: Resource records for the DNS security extensions. *Internet Engineering Task Force*, March 2005. 120

Patrik Arlos and Markus Fiedler. A method to estimate the timestamp accuracy of measurement hardware and software tools. In Steve Uhlig, Konstantina Papagiannaki, and Olivier Bonaventure, editors, *Passive and Active Network Measurement*, volume 4427 of *Lecture Notes in Computer Science*, pages 197–206. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-71616-7. doi: 10.1007/978-3-540-71617-4_20. URL http://dx.doi.org/10.1007/978-3-540-71617-4_20. 43

Grenville Armitage, Mark Claypool, and Philip Branch. *Networking and online games: understanding and engineering multiplayer Internet games*. Wiley. com, 2006. 6

D Awduche, Lou Berger, D Gan, Tony Li, Vijay Srinivasan, and George Swallow. RFC 3209: Extensions to RSVP for LSP Tunnels. *Network Working Group*, 64, 2001. 9, 20

Giacomo Balestra, Salvatore Luciano, Maurizio Pizzonia, and Stefano Vissicchio. Leveraging router programmability for traffic matrix computation. In *Proc. of PRESTO workshop*, 2010. 55

Jeffrey R. Ballard, Ian Rae, and Aditya 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, Berkeley, CA, USA, 2010. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1863133.1863141. 35

T. Bates, E. Chen, and R. Chandra. RFC 4456: BGP Route Reflection: An Alternative to Full Mesh Internal BGP (IBGP). http://tools.ietf.org/html/rfc4456, April 2006. 36

Richard Bellman. On a routing problem. Technical report, DTIC Document, 1956. 20

Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Microte: fine grained traffic engineering for data centers. In *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies*, CoNEXT '11, pages 8:1–8:12, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-1041-3. doi: 10.1145/2079296.2079304. URL http://doi.acm.org/10.1145/2079296.2079304. 38

Bolt Beranek. A history of the arpanet: The first decade. http://www.darpa.mil/WorkArea/DownloadAsset.aspx?id=2677, 1981. 4

Gary Berger. NodeFlow: An OpenFlow Controller Node Style. http://garyberger.net/?p=537, 2012. 30, 33

A. K. Bhushan. RFC 354: File transfer protocol, July 1972. 4

A. K. Bhushan, K. T. Pogran, R. S. Tomlinson, and J. E. White. RFC 561: Standardizing network mail headers, September 1973. 4

A. Bianco, R. Birke, L. Giraudo, and M. Palacin. Openflow switching: Data plane performance. In *IEEE ICC*, may 2010. 47

S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. RFC 2475: An architecture for differentiated services, December 1998. 9

Stewart Brand. *How Buildings Learn: What Happens After They're Built*. Penguin, 1995. 76

British Telecoms. Broadband usage policy. http://bt.custhelp.com/app/answers/detail/a_id/10495/~/broadband-usage-policy, 2013. 97

Broadcom. Bradcom switching silicon chips, 2013. URL http://www.broadcom.com/products/Switching/Data-Center. 16

Martin A. Brown. Pakistan hijacks YouTube. http://www.renesys.com/blog/2008/02/pakistan_hijacks_youtube_1.shtml, 2008. 21

Pat Brundell, Andy Crabtree, Richard Mortier, Tom Rodden, Paul Tennent, and Peter Tolmie. The network from above and below. In *Proc. ACM SIGCOMM W-MUST*, 2011. 78

Michael Butkiewicz, Harsha V. Madhyastha, and Vyas 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, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-1013-0. doi: 10.1145/2068816.2068846. URL http://doi.acm.org/10.1145/2068816.2068846. 6

Z Cai, AL Cox, and TS Eugene Ng. Maestro: balancing fairness, latency and throughput in the openflow control plane. Technical report, Rice University Technical Report TR11-07, 2011. 33, 65

Kenneth L. Calvert, W. Keith Edwards, Nick Feamster, Rebecca E. Grinter, Ye Deng, and Xuzi Zhou. Instrumenting home networks. In *Proceedings of the 2010 ACM SIGCOMM workshop on Home networks*, HomeNets '10, pages 55–60, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0198-5. doi: 10.1145/1851307.1851321. URL http://doi.acm.org/10.1145/1851307.1851321. 38

Marco Canini, Daniele Venzano, Peter Perešíni, Dejan Kostić, and Jennifer 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, Berkeley, CA, USA, 2012. USENIX Association. URL http://dl.acm.org/citation.cfm?id=2228298.2228312. 38

Brian Carpenter and Scott Brim. Rfc 3234: Middleboxes: Taxonomy and issues. 2002. 121

M. Casado, M. Freedman, J. Pettit, N. McKeown, and S. Shenker. Ethane: Taking control of the enterprise. In *Proc. ACM SIGCOMM*, 2007. 35

Cbench. Cbench: controller benchmark, 2010. URL http://docs.projectfloodlight.org/display/floodlightcontroller/Cbench. 58, 65

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–, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7695-1564-9. URL http://dl.acm.org/citation.cfm?id=874028.874134. 23

Stuart Cheshire and Marc Krochmal. Rfc 6762: Multicast dns. *Work in Progress*, 2011a. 122

Stuart Cheshire and Marc Krochmal. Rfc 6763: Dns-based service discovery. 2011b. 122, 126

Marshini Chetty, Richard Banks, Richard Harper, Tim Regan, Abigail Sellen, Christos Gkantsidis, Thomas Karagiannis, and Peter Key. Who's hogging the bandwidth: the consequences of revealing the invisible in the home. In *Proceed-*

ings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '10, pages 659–668, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-929-9. doi: 10.1145/1753326.1753423. URL http://doi.acm.org/10.1145/1753326.1753423. 76, 78, 100

Kenjiro Cho, Kensuke Fukuda, Hiroshi Esaki, and Akira Kato. The impact and implications of the growth in residential user-to-user traffic. *ACM SIGCOMM Computer Communication Review*, 36(4):207, August 2006. ISSN 01464833. doi: 10.1145/1151659.1159938. URL http://portal.acm.org/citation.cfm?doid=1151659.1159938. 75

Lucas Di Cioccio, Rennate Teixeira, and Catherine Rosenberg. Characterizing home networks with homenet profiler. Technical report, Technicolor, 2011. 74, 75

Cisco. Cisco express forwarding overview. http://www.cisco.com/en/US/docs/ios/12_2/switch/configuration/guide/xcfcef.html, May 2005. 18

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. 19

Cisco. Cisco asr 9000 series ethernet line cards. http://www.cisco.com/en/US/prod/collateral/routers/ps9853/data_sheet_c78-501338.html, 2012b. 99

Cisco. Per VLAN Spanning Tree (PVST). http://www.cisco.com/en/US/tech/tk389/tk621/tk846/tsd_technology_support_sub-protocol_home.html, 2012c. 19

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. 19

I Cisco. Cisco Visual Networking Index: Forecast and Methodology, 2011–2016. *CISCO White paper*, 2012e. 7

D. Clark. The design philosophy of the darpa internet protocols. *SIGCOMM Comput. Commun. Rev.*, 18(4):106–114, August 1988. ISSN 0146-4833. doi: 10.1145/52325.52336. URL http://doi.acm.org/10.1145/52325.52336. 4

D. Cohen. RFC 741: Specifications for the network voice protocol (NVP), November 1977. 4

R.D. Corin, M. Gerola, R. Riggio, F. De Pellegrini, and E. Salvadori. Vertigo: Network virtualization and beyond. In *Software Defined Networking (EWSDN), 2012 European Workshop on*, pages 24–29, 2012. doi: 10.1109/EWSDN.2012.19. 34

G.A. Covington, G. Gibb, J.W. Lockwood, and N. Mckeown. A packet generator on the NetFPGA platform. In *Field Programmable Custom Computing Machines, 2009. FCCM '09. 17th IEEE Symposium on*, April 2009. doi: 10.1109/FCCM.2009.29. 45

A. Crabtree, T. Rodden, T. Hemmings, and S. Benford. Finding a place for ubicomp in the home. In *Proc. UbiComp*, 2003. 79

Simon Crosby, Sean Rooney, Rebecca Isaacs, and Herbert Bos. A perspective on how atm lost control. *SIGCOMM Comput. Commun. Rev.*, 32(5):25–28, November 2002. ISSN 0146-4833. doi: 10.1145/774749.774754. URL http://doi.acm.org/10.1145/774749.774754. 26

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. ISSN 0733-8716. doi: 10.1109/49.917713. 24

DARPA. Active Networks. http://www.sds.lcs.mit.edu/darpa-activenet/, April 1997. 22

Jeffrey Dean and Luiz André Barroso. The tail at scale. *Commun. ACM*, 56(2):74–80, February 2013. ISSN 0001-0782. doi: 10.1145/2408776.2408794. URL http://doi.acm.org/10.1145/2408776.2408794. 108

D.S. Decasper, B. Plattner, G.M. Parulkar, Sumi Choi, J.D. DeHart, and T. Wolf. A scalable high-performance active network node. *Network, IEEE*, 13(1):8–19, 1999. ISSN 0890-8044. doi: 10.1109/65.750445. 24

CNET News Declan McCullagh. Dropbox confirms security glitch–no password required. http://cnet.co/kvVbpz, 2011. 109

S. Deering and R. Hinden. RFC 1883: Internet Protocol, version 6 (IPv6) specification, December 1995. 119

E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959. ISSN 0029-599X. doi: 10.1007/BF01386390. URL http://dx.doi.org/10.1007/BF01386390. 20

Roger Dingledine and Nick Mathewson. Design of a blocking-resistant anonymity system. *The Tor Project, Technical Report*, 11:15–16, 2006. 125

Marcel Dischinger, Andreas Haeberlen, Krishna P. Gummadi, and Stefan Saroiu. Characterizing residential broadband networks. *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement - IMC '07*, page 43, 2007a. doi: 10.1145/1298306.1298313. URL http://portal.acm.org/citation.cfm?doid=1298306.1298313. 74

Marcel Dischinger, Andreas Haeberlen, Krishna P Gummadi, and Stefan Saroiu. Characterizing residential broadband networks. In *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. ACM Request Permissions, October 2007b. 96

R. Droms. Dynamic Host Configuration Protocol. RFC 2131, IETF, March 1997. 84

Dmitry Drutskoy, Eric Keller, and Jennifer Rexford. Scalable network virtualization in software-defined networks. *Internet Computing, IEEE*, 17(2):20–27, 2013. ISSN 1089-7801. doi: 10.1109/MIC.2012.144. 34

DSLreports. What is powerboost? http://www.dslreports.com/faq/14520, January 2013. 74

Matthieu Pélissié du Rausas, James Manyika, Eric Hazan, Jacques Bughin, Michael Chui, and Rémi Said. Internet matters: The Net's sweeping impact on growth, jobs, and prosperity. pages 1–13, May 2011. http://www.mckinsey.com/insights/mgi/research/technology_and_innovation/internet_matters. 4

Donald E Eastlake. Rfc 2931: Dns request and transaction signatures (sig (0) s). 2000. 121

L. Eggert, P. Sarolahti, D. Denis-Courmont, V. Stirbu, and H. Tschofenig. A survey of protocols to control network address translators and firewalls. Informational, 2008. 10

Emulab. Emulab: Network emulation testbed, 2000. URL http://www.emulab.net. 58

Endace. Endace DAG Cards. http://www.endace.com/endace-dag-high-speed-packet-capture-cards.html, 2012. 45

David Erickson, Brandon Heller, Shuang Yang, Jonathan Chu, Jonathan Ellithorpe, Scott Whyte, Stephen Stuart, Nick McKeown, Guru Parulkar, and Mendel Rosenblum. Optimizing a virtualized data center. *SIGCOMM Comput. Commun. Rev.*, 41(4):478–479, August 2011. ISSN 0146-4833. doi: 10.1145/2043164.2018530. URL http://doi.acm.org/10.1145/2043164.2018530. 39

Jeffrey Erman, Alexandre Gerber, and Subhabrata Sen. HTTP in the Home : It is not just about PCs. *ACM SIGCOMM Computer Communication ...*, pages 43–48, 2011. URL http://dl.acm.org/citation.cfm?id=1925876. 75

Guy Even, Moti Medina, Gregor Schaffrath, and Stefan Schmid. Competitive and deterministic embeddings of virtual networks. In *Proceedings of 13th International Conference on Distributed Computing and Networking (ICDCN '12)*, January 2012. URL http://www.net.t-labs.tu-berlin.de/papers/EMSS-CDEVN-12.pdf. 38

Federal Communications Commission. A Report on Consumer Wireline Broadband Performance in the U.S. http://www.fcc.gov/measuring-broadband-america/2012/july, 2012. 75

Alessandro Finamore, Marco Mellia, Maurizio M. Munafò, Ruben Torres, and Sanjay 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, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-1013-0. doi: 10.1145/2068816.2068849. URL http://doi.acm.org/10.1145/2068816.2068849. 6

Nate Foster, Rob Harrison, Michael J. Freedman, Christopher Monsanto, Jennifer Rexford, Alec Story, and David Walker. Frenetic: a network programming language. *SIGPLAN Not.*, 46(9):279–291, September 2011. ISSN 0362-1340. doi: 10.1145/2034574.2034812. URL http://doi.acm.org/10.1145/2034574.2034812. 37

Open Network Foundations. https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf, 2012. 26

Global Environment for Network Innovations (GENI). GENI: exploring networks of the future. http://www.geni.net, 2013. 39

Google. OpenFlow @ Google. http://www.opennetsummit.org/archives/apr12/hoelzle-tue-openflow.pdf, 2013. 27

Glenn Greenwald and Ewen 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. 109

R.E. Grinter and W.K. Edwards. The work to make the home network work. In *Proc. ECSCW*, 2005. 8, 76

Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. Nox: towards an operating system for networks. *SIGCOMM Comput. Commun. Rev.*, July 2008a. 43

Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. Nox: towards an operating system for networks. *ACM CCR*, 2008b. 30, 33, 65, 81

Arjun Guha, Mark Reitblatt, and Nate Foster. Machine-verified network controllers. *SIGPLAN Not.*, 48(6):483–494, June 2013. ISSN 0362-1340. doi: 10.1145/2499370.2462178. URL http://doi.acm.org/10.1145/2499370.2462178. 37

P. Gupta and N. McKeown. Algorithms for packet classification. *Network, IEEE*, 15 (2):24–32, 2001. ISSN 0890-8044. doi: 10.1109/65.912717. 54

Robert W Hahn and Scott Wallsten. The economics of net neutrality. *The Economists' Voice*, 3(6), 2006. 96

N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, and R. Johari. Plug-n-Serve: Load-Balancing Web Traffic using OpenFlow. In *ACM SIGCOMM Demo*, August 2009a. 42, 57

Nikhil Handigol, Srinivasan Seetharaman, Mario Flajslik, Nick McKeown, and Ramesh Johari. Plug-n-serve: Load-balancing web traffic using openflow. *ACM SIGCOMM Demo*, 2009b. 35

Nikhil Handigol, S Seetharaman, M Flajslik, Ramesh Johari, and Nick McKeown. Aster* x: Load-balancing as a network primitive. In *9th GENI Engineering Conference (Plenary)*, 2010. 35

Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz, and Nick 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, New York, NY, USA, 2012a. ACM. ISBN 978-1-4503-1775-7. doi: 10.1145/2413176.2413206. URL http://doi.acm.org/10.1145/2413176.2413206. 39

Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Maziéres, and Nick McKeown. Where is the debugger for my software-defined network? In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 55–60, New York, NY, USA, 2012b. ACM. ISBN 978-1-4503-1477-0. doi: 10.1145/2342441.2342453. URL http://doi.acm.org/10.1145/2342441.2342453. 38

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. ISSN 0018-9162. doi: 10.1109/2.755005. 23

T. Hastings, R. Herriot, R. deBry, S. Isaacson, and P. Powell. Rfc 2911: Internet printing protocol/1.1: Model and semantics. September 2000. 106

Seppo Hätönen, Aki Nyrhinen, Lars Eggert, Stephen Strowes, Pasi Sarolahti, and Markku Kojo. An experimental study of home gateway characteristics. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, IMC '10, pages 260–266, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0483-2. doi: 10.1145/1879141.1879174. URL http://doi.acm.org/10.1145/1879141.1879174. 75

Brandon Heller, Srini Seetharaman, Priya Mahadevan, Yiannis Yiakoumis, Puneet Sharma, Sujata Banerjee, and Nick 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, Berkeley, CA, USA, 2010. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1855711.1855728. 37

Thomas R. Henderson, Sumit Roy, Sally Floyd, and George F. Riley. ns-3 project goals. In *Proceeding from the 2006 workshop on ns-2: the IP network simulator*, WNS2 '06, New York, NY, USA, 2006. ACM. ISBN 1-59593-508-8. doi: 10.1145/1190455.1190468. URL http://doi.acm.org/10.1145/1190455.1190468. 59, 63

M. Hicks, J.T. Moore, D.S. Alexander, C.A. Gunter, and S.M. Nettles. Planet: an active internetwork. In *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1124–1133 vol.3, 1999. doi: 10.1109/INFCOM.1999.751668. 23

Michael Hicks, Pankaj Kakkar, Jonathan T. Moore, Carl A. Gunter, and Scott Nettles. Plan: a packet language for active networks. *SIGPLAN Not.*, 34(1):86–93, September 1998. ISSN 0362-1340. doi: 10.1145/291251.289431. URL http://doi.acm.org/10.1145/291251.289431. 23

Michio Honda, Yoshifumi Nishida, Costin Raiciu, Adam Greenhalgh, Mark Handley, and Hideyuki Tokuda. Is it still possible to extend TCP? In *IMC '11: Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. ACM Request Permissions, November 2011. 10

C. Hopps. Rfc 2992: Analysis of an equal-cost multi-path algorithm. 2000. 32, 38

HP. Provision^{TM}asic: Built for the future. http://www.hp.com/rnd/itmgrnews/built_for_future.htm, January 2012. 16

Te-Yuan Huang, Kok-Kiong Yap, Ben Dodson, Monica S. Lam, and Nick McKeown. Phonenet: a phone-to-phone network for group communication within an administrative domain. In *Proceedings of the second ACM SIGCOMM workshop on Networking, systems, and applications on mobile handhelds*, MobiHeld '10, pages 27–32, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0197-8. doi: 10.1145/1851322.1851331. URL http://doi.acm.org/10.1145/1851322.1851331. 39

IEEE. IEEE Std 802.1D-1998: Media Access Control (MAC) Bridges. http://standards.ieee.org/getieee802/download/802.1D-1998.pdf, 1998. 19

IEEE. IEEE Std 802.1D-2004: Media Access Control (MAC) Bridges. http://standards.ieee.org/getieee802/download/802.1D-2004.pdf, 2004. 19

IEEE. IEEE Std 802.1Q: Virtual Bridged Local Area Networks. http://www.dcs.gla.ac.uk/~lewis/teaching/802.1Q-2005.pdf, 2005. 19

IEEE. Ieee 802.1 aq shortest path bridging. http://www.ieee802.org/1/pages/802.1aq.html, 2006. 19

IEEE. Ieee 802.1 ab station and media access control connectivity discovery. http://standards.ieee.org/getieee802/download/802.1AB-2009.pdf, September 2009. 19

Intel. Intel ethernet switching components. URL http://ark.intel.com/products/family/134/Intel-Ethernet-Switching-Components. 16

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. 8

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. 8

Teerawat Issariyakul. *Introduction to network simulator NS2*. Springer Science+ Business Media, 2012. 58

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 *Software Defined Networking (EWSDN), 2012 European Workshop on*, pages 48–53, 2012. doi: 10.1109/EWSDN.2012.15. 44

Lavanya Jose, Minlan Yu, and Jennifer Rexford. Online measurement of large traffic aggregates on commodity switches. In *Proc. of the USENIX HotICE workshop*, 2011. 55

IDG News Service Juan Carlos Perez. Facebook's beacon more intrusive than previously thought. http://www.pcworld.com/article/140182/article.html, November 2007. 108

Juniper. T series core router architecture overview. http://www.slideshare.net/junipernetworks/t-series-core-router-architecture-review-whitepaper, 2012. 18

JupiterResearch 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. 6

Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie 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, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-771-4. doi: 10.1145/1644893.1644918. URL http://doi.acm.org/10.1145/1644893.1644918. 69

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

Ahmed Khurshid, Wenxuan Zhou, Matthew Caesar, and P. Brighten Godfrey. Veriflow: verifying network-wide invariants in real time. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 49–54, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1477-0. doi: 10.1145/2342441.2342452. URL http://doi.acm.org/10.1145/2342441.2342452. 38

Hyojoon Kim, Theophilus Benson, Aditya Akella, and Nick 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, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-1013-0. doi: 10.1145/2068816.2068863. URL http://doi.acm.org/10.1145/2068816.2068863. 8

Masayoshi Kobayashi, Srini Seetharaman, Guru Parulkar, Guido Appenzeller, Joseph Little, Johan van Reijendam, Paul Weissmann, and Nick McKeown. Maturing of OpenFlow and Software Defined Networking through Deployments. August 2012. 27, 50

Teemu Koponen. Software is the Future of Networking. https://sites.google.com/a/ancsconf.org/2012/downloads/Koponen%20ANCS%202012%20keynote.pdf, 2012. 135

Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, and Scott Shenker. Onix: a distributed control platform for large-scale production networks. In *Proceedings of the 9th USENIX conference on Operating systems de-*

sign and implementation, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1924943.1924968. 36

Vasileios Kotronis, Xenofontas Dimitropoulos, and Bernhard 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, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1776-4. doi: 10.1145/2390231.2390241. URL http://doi.acm.org/10.1145/2390231.2390241. 36

Maxim Krasnyansky, Maksim Yevmenkin, and Florian Thiel. Universal tun/-tap device driver. https://www.kernel.org/doc/Documentation/networking/tuntap.txt, 2002. 124

Christian Kreibich, Nicholas Weaver, Boris Nechaev, and Vern Paxson. Netalyzr: illuminating the edge network. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, IMC '10, pages 246–259, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0483-2. doi: 10.1145/1879141.1879173. URL http://doi.acm.org/10.1145/1879141.1879173. 74

Balachander Krishnamurthy and CE Wills. On the leakage of personally identifiable information via online social networks. *...ACM workshop on Online social networks*, 2009. URL http://dl.acm.org/citation.cfm?id=1592668. 108

Nate Kushman, Srikanth Kandula, and Dina Katabi. Can you hear me now?!: it must be bgp. *SIGCOMM Comput. Commun. Rev.*, 37(2):75–84, March 2007. ISSN 0146-4833. doi: 10.1145/1232919.1232927. URL http://doi.acm.org/10.1145/1232919.1232927. 21

A Kuzmanovic, A Mondal, S Floyd, and K Ramakrishnan. Rfc 5562: adding explicit congestion notification (ecn) capability to tcps syn, June 2009. 9

Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM*

Workshop on Hot Topics in Networks, Hotnets-IX, pages 19:1–19:6, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0409-2. doi: 10.1145/1868447.1868466. URL http://doi.acm.org/10.1145/1868447.1868466. 39

Anna Leach. Greedy Sky admits: We crippled broadband with TOO MANY users, 2013. 97

Paul J. Leach and Dilip 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. 106

John T. Lewis, Raymond Russell, Fergal Toomey, Brian McGurk, Simon Crosby, and Ian Leslie. Practical connection admission control for ATM networks based on on-line measurements. *Computer Communications*, 21 (17):1585 – 1596, 1998. ISSN 0140-3664. doi: 10.1016/S0140-3664(98)00224-2. URL http://www.sciencedirect.com/science/article/pii/S0140366498002242. 26

Li Erran Li, Z. Morley Mao, and Jennifer Rexford. Toward software-defined cellular networks. In *Proceedings of the 2012 European Workshop on Software Defined Networking*, EWSDN '12, pages 7–12, Washington, DC, USA, 2012. IEEE Computer Society. ISBN 978-0-7695-4870-8. doi: 10.1109/EWSDN.2012.28. URL http://dx.doi.org/10.1109/EWSDN.2012.28. 39

LXC. LXC Linux Containers. http://lxc.sourceforge.net, 2013. 39

Anil Madhavapeddy, Richard Mortier, Charalampos Rotsos, David Scott, Balraj Singh, Thomas Gazagnaire, Steven Smith, Steven Hand, and Jon Crowcroft. Unikernels: Library operating systems for the cloud. In *Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems*, pages 461–472. ACM, 2013. 42, 64

Ratul Mahajan, David Wetherall, and Tom Anderson. Understanding bgp misconfiguration. *SIGCOMM Comput. Commun. Rev.*, 32(4):3–16, August 2002. ISSN 0146-4833. doi: 10.1145/964725.633027. URL http://doi.acm.org/10.1145/964725.633027. 8

Gregor Maier, A Feldmann, Vern Paxson, and Mark Allman. On dominant characteristics of residential broadband internet traffic. *... conference on Internet ...*, 2009. URL http://dl.acm.org/citation.cfm?id=1644904. 75

G. Malkin. RFC 2453: RIP version 2, November 1998. 20

ML Mazurek, JP Arsenault, and Joanna Bresee. Access control for home data sharing: Attitudes, needs and practices. *Proceedings of the 28th ...*, pages 645–654, 2010. URL http://dl.acm.org/citation.cfm?id=1753421. viii, 77

Nick McKeown, Martin Casado, and Tom Edsall. Cs244: Advanced topics in networking, spring 2012. http://www.stanford.edu/class/cs244/2012/, 2012. 40

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. 24

Microsoft. Windows filtering platform, 2012. URL http://msdn.microsoft.com/en-us/library/windows/desktop/aa366510.aspx. 98

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

CVNI Mobile. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2011–2016 . *White Paper*, 2012. 7

P. V. Mockapetris. RFC 1034: Domain names — concepts and facilities, November 1987. 119

Christopher Monsanto, Nate Foster, Rob Harrison, and David 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, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1083-3. doi: 10.1145/2103656.2103685. URL http://doi.acm.org/10.1145/2103656.2103685. 37

Christopher Monsanto, Joshua Reich, Nate Foster, Jennifer Rexford, and David Walker. Composing software-defined networks. In *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation*, nsdi'13, pages 1–14, Berkeley, CA, USA, 2013. USENIX Association. URL http://dl.acm.org/citation.cfm?id=2482626.2482629. 33, 37

A.B. Montz, D. Mosberger, S.W. O'Mally, L.L. Peterson, and T.A. Proebsting. Scout: a communications-oriented operating system. In *Hot Topics in Operating Systems, 1995. (HotOS-V), Proceedings., Fifth Workshop on*, pages 58–61, 1995. doi: 10.1109/HOTOS.1995.513455. 23

Richard Mortier, Ben Bedwell, Kevin Glover, Tom Lodge, Tom Rodden, Charalampos Rotsos, Andrew W. Moore, Alexandros Koliousis, and Joseph Sventek. Supporting novel home network management interfaces with OpenFlow and NOX. In *Proc. ACM SIGCOMM*, 2011. demo abstract. 81

J. Moy. RFC 2328: OSPF version 2, April 1998. 20

Jad Naous, David Erickson, G. Adam Covington, Guido Appenzeller, and Nick McKeown. Implementing an openflow switch on the netfpga platform. In *ANCS*, 2008. 47

The netfilter.org project. libnetfilter conntrack project, 2010. URL http://www.netfilter.org/projects/libnetfilter_conntrack/index.html. 98

Carrier Ethernet News. Upgrading to bigger backhaul bandwidth for dslams. http://www.carrierethernetnews.com/articles/319137/upgrading-to-bigger-backhaul-bandwidth-for-dslams/, 2011. 95

ofcom. The Consumer Experience of 2012, 2012. URL http://stakeholders.ofcom.org.uk/binaries/research/consumer-experience/tce-12/Consumer_Experience_Researc1.pdf. 75

Ofelia. FP7 Ofelia. http://www.fp7-ofelia.eu/, 2013. 39

OFLOPS. OFLOPS manual, 2011. http://www.openflow.org/wk/index.php/Oflops. 45

R. Olsson. *pktgen*, the linux packet generator. In *Proc. Linux Symposium*, 2005a. 90

R. Olsson. pktgen the linux packet generator. In *Proceedings of Linux symposium*, 2005b. 45

Sean W. O'Malley and Larry L. Peterson. A dynamic network architecture. *ACM Trans. Comput. Syst.*, 10(2):110–143, May 1992. ISSN 0734-2071. doi: 10.1145/128899.128901. URL http://doi.acm.org/10.1145/128899.128901. 22

OpenFlow Consortium. OpenFlow reference implementation. http://www.openflow.org/wp/develop/, 2009a. 47

OpenFlow Consortium. Openflow switch specification (version 1.0.0). www.openflow.org/documents/openflow-spec-v1.0.0.pdf, December 2009b. 46, 48, 52

D. Oran. RFC 1142: OSI IS-IS intra-domain routing protocol, February 1990. 20

Eric D Osborne and Ajay Simha. *Traffic engineering with MPLS*. Cisco Press, 2002. 20

Abhinav Pathak, Ming Zhang, Y. Charlie Hu, Ratul Mahajan, and Dave 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, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-1013-0. doi: 10.1145/2068816.2068859. URL http://doi.acm.org/10.1145/2068816.2068859. 20

Joshua Pelkey and George 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, ICST, Brussels, Belgium, Belgium, 2011. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). ISBN 978-1-936968-00-8. URL http://dl.acm.org/citation.cfm?id=2151054.2151128. 64

Radia Perlman. An algorithm for distributed computation of a spanningtree in an extended lan. *SIGCOMM Comput. Commun. Rev.*, 15(4):44–53, September 1985. ISSN 0146-4833. doi: 10.1145/318951.319004. URL http://doi.acm.org/10.1145/318951.319004. 19

Justin Pettit, Jesse Gross, Ben Pfaff, Martin Casado, and Simon Crosby. Virtualizing the network forwarding plane. In *DC-CAVES*, 2010. 47, 48, 66, 81

PlanetLab. PlanetLab: An open platform for developing, deploying and accessing planetary-scale services, 2007. URL http://www.planet-lab.org. 58

Philip Porras, Seungwon Shin, Vinod Yegneswaran, Martin Fong, Mabry Tyson, and Guofei Gu. A security enforcement kernel for openflow networks. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 121–126, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1477-0. doi: 10.1145/2342441.2342466. URL http://doi.acm.org/10.1145/2342441.2342466. 35

J. Postel. RFC 791: Internet Protocol, September 1981. 10

Johan A Pouwelse, Pawel Garbacki, D Epema, and Henk J Sips. A measurement study of the bittorrent peer-to-peer file-sharing system. Technical report, Technical Report PDS-2004-003, Delft University of Technology, The Netherlands, 2004. 6

Pox. About pox. http://www.noxrepo.org/pox/about-pox/, 2012. 33

Ian Pratt. Pratt's Test TCP. http://www.cl.cam.ac.uk/research/srg/netos/netx/index.html, 2002. 62

Privoxy. Privoxy - home page. http://privoxy.org, 2013. 125

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. 97

DCAN project. DCAN - "Devolved Control of ATM Networks". http://www.cl.cam.ac.uk/research/srg/netos/old-projects/dcan/, 2000. 24

Quagga. Quagga Routing Suite. http://www.quagga.net. 36

A.H. Rasti and R. Rejaie. Understanding peer-level performance in bittorrent: A measurement study. In *Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on*, pages 109–114, 2007. doi: 10.1109/ICCCN.2007.4317805. 6

Ahlem Reggani, Fabian Schneider, and Renata Teixeira. An end-host view on local traffic at home and work. In *Proceedings of the 13th international conference on Passive and Active Measurement*, PAM'12, pages 21–31, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-28536-3. doi: 10.1007/978-3-642-28537-0_3. URL http://dx.doi.org/10.1007/978-3-642-28537-0_3. 75

Mark Reitblatt, Nate Foster, Jennifer Rexford, Cole Schlesinger, and David 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, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1419-0. doi: 10.1145/2342356.2342427. URL http://doi.acm.org/10.1145/2342356.2342427. 37

Y. Rekhter. RFC 1265: BGP protocol analysis, October 1991. 21

E Rescorla. Rfc 2631: Diffie-hellman key agreement method. *Internet Engineering Task Force*, 1999. 121

Return Infinity. Baremetalos. http://www.returninfinity.com/baremetal.html, 2013. 60

T. Richardson and J. Levine. Rfc 6143: The remote framebuffer protocol. *Work in Progress*, March 2011. 106

Tom Rodden and Steve Benford. The evolution of buildings and implications for the design of ubiquitous domestic environments. *Proceedings of the conference on Human factors in computing systems - CHI '03*, (5):9, 2003. doi: 10.1145/642614.642615. URL http://portal.acm.org/citation.cfm?doid=642611.642615. 76

S. Rooney. The hollowman: an innovative atm control architecture. In *Proceedings of the fifth IFIP/IEEE international symposium on Integrated network management V : integrated management in a virtual world: integrated management in a virtual world*, pages 369–380, London, UK, UK, 1997. Chapman & Hall, Ltd. ISBN 0-412-80960-5. URL http://dl.acm.org/citation.cfm?id=280039.280084. 25

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. ISSN 0163-6804. doi: 10.1109/35.722136. 25

E. Rosen, A. Viswanathan, and R. Callon. Rfc 3031: Multiprotocol label switching architecture. *MPLS Working Group*, 64, 2001. 19

Christian Esteve Rothenberg, Marcelo Ribeiro Nascimento, Marcos Rogerio Salvador, Carlos Nilton Araujo Corrêa, Sidney Cunha de Lucena, and Robert Raszuk. Revisiting routing control platforms with the eyes and muscles of software-defined networking. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 13–18, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1477-0. doi: 10.1145/2342441.2342445. URL http://doi.acm.org/10.1145/2342441.2342445. 36

Charles L. Rutgers. 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. 21

Dipjyoti Saikia. Mul OpenFlow controller. http://sourceforge.net/projects/mul/, 2013. 33

Nadi Sarrar, Steve Uhlig, Anja Feldmann, Rob Sherwood, and Xin Huang. Leveraging zipf's law for traffic offloading. *SIGCOMM Comput. Commun. Rev.*, 42(1):16–22, January 2012. ISSN 0146-4833. doi: 10.1145/2096149.2096152. URL http://doi.acm.org/10.1145/2096149.2096152. 36

Beverly Schwartz, Alden W. Jackson, W. Timothy Strayer, Wenyi Zhou, R. Dennis Rockwell, and Craig Partridge. Smart packets: applying active networks to network

management. *ACM Trans. Comput. Syst.*, 18(1):67–88, February 2000. ISSN 0734-2071. doi: 10.1145/332799.332893. URL http://doi.acm.org/10.1145/332799.332893. 22

Aman Shaikh and Albert Greenberg. Experience in black-box ospf measurement. In *ACM IMC*, 2001. 52

E. Shehan and W.K. Edwards. Home networking and HCI: What hath God wrought? In *Proc. ACM CHI*, 2007. 77

E. Shehan-Poole, M. Chetty, W.K. Edwards, and R.E. Grinter. Designing interactive home network maintenance tools. In *Proc. ACM DIS*, 2008. 76, 77

S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck. Rfc3530: Network file system (nfs) version 4 protocol. April 2003. 106

S. Shepler, M. Eisler, and D. Noveck. Rfc 5661: Network file system (nfs) version 4 minor version 1 protocol, June 2010. 130

Rob Sherwood, Michael Chan, Adam Covington, Glen Gibb, Mario Flajslik, Nikhil Handigol, Te-Yuan Huang, Peyman Kazemian, Masayoshi Kobayashi, Jad Naous, Srinivasan Seetharaman, David Underhill, Tatsuya Yabe, Kok-Kiong Yap, Yiannis Yiakoumis, Hongyi Zeng, Guido Appenzeller, Ramesh Johari, Nick McKeown, and Guru Parulkar. Carving research slices out of your production networks with openflow. *SIGCOMM Comput. Commun. Rev.*, 40:129–130, January 2010a. ISSN 0146-4833. doi: http://doi.acm.org/10.1145/1672308.1672333. URL http://doi.acm.org/10.1145/1672308.1672333. 34

Rob Sherwood, Glen Gibb, Kok-Kiong Yapa, Martin Cassado, Guido Appenzeller, Nick McKeown, and Guru Parulkar. Can the production network be the test-bed? In *OSDI*, 2010b. 42, 99

Kai-Yeung Siu and Raj Jain. A brief overview of atm: protocol layers, lan emulation, and traffic management. *SIGCOMM Comput. Commun. Rev.*, 25(2):6–20, April 1995. ISSN 0146-4833. doi: 10.1145/210613.210616. URL http://doi.acm.org/10.1145/210613.210616. 8

Srikanth Sundaresan and W de Donato. Broadband internet performance: a view from the gateway. *SIGCOMM-Computer* . . . , (Section 5), 2011. URL http://wpage.unina.it/walter.dedonato/pubs/bb_sigcomm11.pdf. 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 *Proc. IM*, 2011. 80, 82

Big Switch. Floodlight OpenFlow Controller. http://www.projectfloodlight.org/floodlight/, 2013. 30, 33, 44

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

Ajay Tirumala, Feng Qin, Jon Dugan, Jim Ferguson, and Kevin Gibbs. Iperf: The tcp/udp bandwidth measurement tool. 2005. 102

P. Tolmie, A. Crabtree, T. Rodden, C. Greenhalgh, and S. Benford. Making the home network at home: digital housekeeping. In *Proceedings ECSCW*, 2007. 76, 77, 79

Amin Tootoonchian, Monia Ghobadi, and Yashar Ganjali. OpenTM: traffic matrix estimator for openflow networks. In *PAM*, 2010. 55

Trema. Trema: Full-Stack OpenFlow Framework in Ruby and C. http://trema.github.io/trema/, 2011. 33

Jacobus Erasmus van der Merwe. Open service support for ATM. Technical Report UCAM-CL-TR-450, University of Cambridge, Computer Laboratory, 15 JJ Thomson Avenue, Cambridge CB3 0FD, United Kingdom, phone +44 1223 763500, November 1998. v, 25

András Varga and Rudolf 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, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute

for Computer Sciences, Social-Informatics and Telecommunications Engineering). ISBN 978-963-9799-20-2. URL http://dl.acm.org/citation.cfm?id=1416222.1416290. 58

Virgin Media. Virgin media cable traffic management policy. http://help.virginmedia.com/system/selfservice.controller?CMD=VIEW_ARTICLE&ARTICLE_ID=2781&CURRENT_CMD=SEARCH&CONFIGURATION=1002&PARTITION_ID=1&USERTYPE=1&LANGUAGE=en&COUNTY=us&VM_CUSTOMER_TYPE=Cable. 97

Andreas Voellmy, Ashish Agarwal, and Paul Hudak. Nettle: Functional reactive programming for openflow networks. Technical Report YALEU/DCS/RR-1431, Yale University, July 2010. 33

Andreas Voellmy, Hyojoon Kim, and Nick Feamster. Procera: a language for high-level reactive network control. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 43–48, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1477-0. doi: 10.1145/2342441.2342451. URL http://doi.acm.org/10.1145/2342441.2342451. 37

Richard Wang, Dana Butnariu, and Jennifer 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, Berkeley, CA, USA, 2011. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1972422.1972438. 35

David Watson, Farnam Jahanian, and Craig 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–, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1920-2. URL http://dl.acm.org/citation.cfm?id=850929.851939. 21

Paul Weissmann and Srini 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. 42

D.J. Wetherall, John 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. 22

Wikipedia. Network performance. http://en.wikipedia.org/wiki/Network_performance, 2014. 5

Mike P. Wittie, Veljko Pejovic, Lara Deek, Kevin C. Almeroth, and Ben 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, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0448-1. doi: 10.1145/1921168.1921201. URL http://doi.acm.org/10.1145/1921168.1921201. 108

Andreas Wundsam, Dan Levin, Srini Seetharaman, and Anja 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, Berkeley, CA, USA, 2011. USENIX Association. URL http://dl.acm.org/citation.cfm?id=2002181.2002210. 38

Xen Project. XAPI. http://www.xenproject.org/developers/teams/xapi.html. 63

Kok-Kiong Yap, Masayoshi Kobayashi, David Underhill, Srinivasan Seetharaman, Peyman Kazemian, and Nick McKeown. The stanford openroads deployment. In *Proceedings of ACM WINTECH*, 2009. 39, 57

Kok-Kiong Yap, Rob Sherwood, Masayoshi Kobayashi, Te-Yuan Huang, Michael Chan, Nikhil Handigol, Nick McKeown, and Guru Parulkar. Blueprint for introducing innovation into wireless mobile networks. In *Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures*, VISA '10, pages 25–32, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0199-2. doi: 10.1145/1851399.1851404. URL http://doi.acm.org/10.1145/1851399.1851404. 39

Mark Yarvis, Konstantina Papagiannaki, and W. Steven Conner. Characterization of 802.11 wireless networks in the home. In *In Proceedings of the 1st workshop on Wireless Network Measurements (Winmee)*, 2005. 74

Yiannis Yiakoumis, Kok-Kiong Yap, Sachin Katti, Guru Parulkar, and Nick McKeown. Slicing home networks. In *Proceedings of the 2nd ACM SIGCOMM workshop on Home networks*, HomeNets '11, pages 1–6, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0798-7. doi: 10.1145/2018567.2018569. URL http://doi.acm.org/10.1145/2018567.2018569. 39

Yiannis Yiakoumis, Sachin Katti, Te-Yuan Huang, Nick McKeown, Kok-Kiong Yap, and Ramesh Johari. Putting home users in charge of their network. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, pages 1114–1119, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1224-0. doi: 10.1145/2370216.2370451. URL http://doi.acm.org/10.1145/2370216.2370451. 39

T. Ylonen and Ed. C. Lonvick. RFC 4235 : The Secure Shell (SSH) Transport Layer Protocol, January 2006. 106

Minlan Yu, Jennifer Rexford, Michael J. Freedman, and Jia Wang. Scalable flow-based networking with difane. *SIGCOMM Comput. Commun. Rev.*, 41(4):–, August 2010a. ISSN 0146-4833. URL http://dl.acm.org/citation.cfm?id=2043164.1851224. 36

Minlan Yu, Jennifer Rexford, Michael J. Freedman, and Jia Wang. Scalable flow-based networking with difane. In *ACM SIGCOMM*, August 2010b. 42