Flexible Software Defined Network

Charalampos Rotsos

Computer Laboratory

University of Cambridge

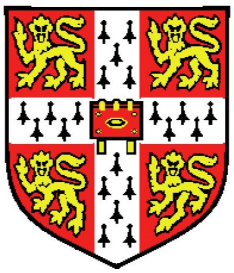
A thesis submitted for the degree of

Doctor of Philosophy

Yet to be decided

Abstract

Flexible Software Defined Network



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

4, add a reference to the value of the cloud industry.

6, find references for OSI TP* protocol and ATM UNI

12, A bit strange. need to rephrase

36, Add latest results

38, Maybe remove this section

39, Add some notes on SDNSIM

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

73, describe some efforts to overcome middleboxes.

73, IPv6 can do some of that,but not everything

76, Add dropbox measurement study

76, report Dropbox problem

77, Add some reference to cloud controller

82, Mention that tactic is able to control OpenFlow also

82, Discuss weight of tactics

87, discnnected Signpost functionality

89, Mention OpenVPN ARP cache

Chapter 1

Introduction

Internet has become the predominant mode of communication of our times. Currently, 1/3 of earth population is connected to the Internet ITU [2011], while Internet-related business is estimated to account for 3.4% of the global GDP du Rausas et al. [2011]. In paraller, a large fraction of our everyday social life requires network/Internet connectivity. While computer networks play an important role in our everyday life, their strong backwards compatibility requirement create a important gap in their functionality evolution in order to fulfil current evolving communication needs. As a result, current network technologies are not able to fulfil the novel functional requirements of the social setting, while providing continuous connectivity.

Our work focuses on the evolvability problem of modern networks. The key idea of this work investigates new network control mechanisms that evolve functionality and can scale for large network sizes. In this dissertation we argue the thesis that:

> Computer network design should compat the problem of network ossification through context-aware evolved control planes, in order to provide new functionalities to the inter-connecting fabric. Such control mechanisms should address the requirements of the deployment environment and establish new domain-specialized control abstractions that take advantage of its distinct properties.

For the remainder of this introduction we justify the importance of this thesis. In Section 1.1 we present in details the limitations of current Internet design and the inherent evolutionary difficulties of its protocols. In Section 1.2, we list our contributions

Application	rate	latency	jitter	# connections
web	0	0	0	0
video	0	0	0	0
p2p	0	0	0	0
voip	0	0	0	0
game	0	0	0	0

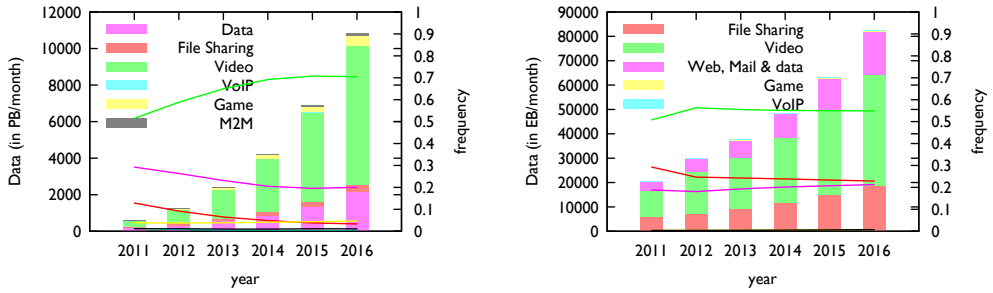
Table 1.1: Network performance requirement for a set of popular traffic classes.

and in Section 1.3, we present the content of each chapter of the thesis. Finally, in section 1.4 we list the publications relating to the content of this thesis.

1.1 Motivation

rt

Computer network evolution



(a) global mobile traffic trend

(b) global Internet traffic trend

Figure 1.1: Cisco Visual Network Index reports on global network traffic per application. Subfigure 1.1(a) provides details on the global Internet traffic trends, while Subfigure 1.1(b) focuses on Mobile Internet traffic.

One of the key mechanisms that formed the objective conditions for the digital revolution of our era, was the concept of computer networking. Initially, Computer networks aimed to provide a new communication architecture that would allow continuous communication over a redundant network, even when a significant number of

links was destroyed. The main building block of computer networks is the idea of packet-switched networks Licklider [1963]. This idea gave birth to the pioneer of today's Internet, the *ARPANET* Mills and Braun [1987], allowing for the first time in computing history communication between multiple computers over a mess network. The initial set of applications that were standardised where : e-mail Bhushan et al. [1973], ftp Bhushan [1972] and voice Cohen [1977]. This initial implementation was later replaced by the NSFNET in the 80's, which finally devolved in today's Internet. As part of this transition, the research community developed also the standards for the TCP/IP protocol suite Clark [1988], the default protocol to provide connectivity for the Internet.

Since the time of the ARPANET, computer networks have seen a significant elevation on their role in the social apparatus of our world due to a number of reasons. One of the most important trends, that increased their usage, was the radical reduction in cost and size of network-enabled personal computers, following Moore's Law model. The low cost factor of personal computers along with the highly programmable nature of their CPU, made them an elegant platform to develop applications that introduce new functionalities seamlessly. Nowadays, programmable CPUs are integrated in a number of multi-purpose devices such as mobile phones, display devices etc, while the ability of personal computers to transform in size, introduces new computing concepts, such as laptops, tablets and other. As a result, the paradigm of one computer per household of the 90's rapidly shifted to the paradigm of multiple devices per user, replacing a number of everyday single-purpose devices Dholakia [2006]. On one hand, this augmentation in computational devices drives to a great extend the development in network technologies in providing new inte-device communication mechanisms. A number of network-enabled applications are developed that addresses these requirements, and new network paradigms are introduced to accomodate these changes, like home networks and hotspots. On the other hand, the elevated role of computer networks and the introduction of the cloud computing paradigm, introduce a number of internet-wide services with a global scope. The important role of computer networks for the society can be further reflected in the government level debate to proclaim Internet connectivity as a fundamental human right Klang and Murray [2005].

In parallel with the development of personal networking, computer network have become widely adopted as an integral asset for the enterprise world. Currently the In-

ternet produces 4,3% of the global GDP. Computer Networking and the Internet, provide the middleware to interconnect modern multinational businesses. In the business domain computer network have become popular and important for two main reasons: computer networks provide a cheap and fast communication medium to interconnect the business logic, and distribute content to users. The adaptation of computer network has further augmented through the utilisation of the cloud as a medium to offload infrastructures to 3rd party cloud providers, reducing to a great extend the cost of running services in house. *add a reference to the value of the cloud industry.*

The wide adaptation of computer networks has introduce a number of new use cases and applications that introduce significant performance requirements in the short scale. Computer networking depends to a great extend on the abstraction design pattern in order to support scalability and heterogeneity. The abstraction principle is based to a great extend on the OSI model Day and Zimmermann [1983], which tries to separate network functionality into a number of layers and define the interface provided by each layer. A side effect of this design is that application developers remain agnostic to lower layer intternal state. Further, The OSI model and the TCP/IP protocol suite luck performance-related semantics in their interface definitions. Applications on the other hand, tend to behave egotistically, and aggrsively try to fulfil performance requirements in the deployed environment. As a result, short-scale resource allocation in a network becomes difficult due to the diverse nature of network applications. In order to exemplify the problem, we list in Table 1.1 a number of key traffic properties for popular traffic classes of the Internet. Network applications have diverse properties which becomes difficult to address as link utilisation increases.

Diversity in network application requirements hardens also network planning. The main cause of this problem is the high churn in the popularity of network applications. In order to exhibit this trend, we plot in Figure 1.1 the global prediction on traffic volumes for popular application classes for five years. We use data from cisco visualization index white papers Cisco [2012]; Mobile [2012]. In the histogram we can see that network traffic is expected to increase an order of magnitude for the mobile environment, while the global Internet traffic is expected to increase four times. In parallel, application volumes evolve unevenly between traffic classes. File sharing services are expected to reduce their share of the total volume, replaced by web and video delivery services.

High diversity is also observed on the properties of available mediums for computer networks. The properties of links is defined in the data link and physical layer of the OSI model. Currently, Ethernet is the predominant link layer protocol in the Internet. In the 80s the low cost property of Ethernet implementations establish it as the leader of the market ever since. The protocol has developed standards to run over coper and optical mediums, as well as off-licence radio frequencies, satellite and mobile networks. Although the Ethernet abstraction is persistent among all these mediums, it hides a lot of the performance limitations of the link (e.g. packet loss, hop-by-hop ARQ etc.). Because of this diversity in links, the performance of a computer network can be variable. An example of this property is the Internet. Internet exhibits a 3 layer hierarchy of ASes, which allows it to scale and provide short-length paths between any 2 nodes. Tier 1 and 2 ISPs provide forwarding in an homogeneous and fast manner. Such ISP's are in charge of a relatively small number of network points and thus are able to upgrade network infrastructure with relatively low costs, which can further be offloaded to clients through SLA agreements. For Tier-3 ISPs things are a lot different. This class of ISPs covers a wide range of services. Also because this is the last hop to end users, such networks tend to be large and spread over large geographic distances. For this type of networks, connectivity properties are variable, users SLA have minimum guarantees, performance can be highly dependent on link sharing ratio and can be highly variant due to the heterogeneity of medium types. Additionally, the cost to upgrade such networks is high, while strong market competition makes difficult to offload costs directly to users. A number of measurement studies have described these differences Dischinger et al. [2007b]; Huang et al. [2010].

Computer network ossification

Although computer networks are highly important for society the adaptability of network technologies to user requirement has not been equal over the years. This mismatch can be ascribed to a number of reasons.

Current network technologies were developed a number of years ago in order to develop standardized and generic mechanisms to interconnect research institutes. Although DARPA funded the idea of computer networks in order to develop new resilient communication mechanisms, the early adopters of the technologies were universities

and research facilities. As a result, protocols were developed by computer scientist taking under consideration the properties of such environments. The TCP/IP protocol suite was develop during the transition of the ARPANET to NSFNET. Since then, the TCP/IP protocol suite has been the default standard of the Internet. During the first period of the NSFNET, a number of competitive suites were developed which addressed in their specification the problem of extensibility *find references for OSI TP* protocol and ATM UNI* . Unfortunately, the increased design complexity made it difficult to develop high performance implementations, and they soon were declared obsolete by the network community. The TCP/IP protocol suite provided a fair split between simplicity and extensibility at that time.

Nontheless, in the recent years the limitations of TCP/IP abstraction have become apparent, as a number of fundamental assumptions has changed. Some of the core limitations of the protocol can be described in the following points:

Elevated Role of Security : An important architectural goal for the design of computer network was the minimization of functional requirements from joining hosts, allowing wide adoption of the technology and open accessibility. When the idea of computer networks was first developed, the capabilities of computer hardware were limited and network connectivity amied to minimize the computational requirements. As a result, the initial security concerns of computer network technologies were minimized. In the recent years, due to the vital role of computer networks in industry, security requirements expanded. A McAfee report from 2009 reports that the cost of cybersecurity is calculated to approximately six hundred million dollars Kanan et al. [2009]. The threat model lurcking over the Internet is wide and contains a number of threats, from Information interception to denial-of-service attacks. Such costs can be reduced to a great extend if the security was inherent to network protocols, span from the lowest levels of the network abstraction and spread across the network. Attempts to address such problem have been proposed in the protocol community, e.g. IPSEC Kent and Atkinson [1998], but the deployment at the moment is not straighforward.

Network Addressing : When the IP protocol was firstly deployed in the Internet, the size of the network was sufficiently small. Addressing was assigned based on a 32-bit integer space, split in byte aligned classes in order to permit aggregation at the

forwarding entities. Within 10 years, the initial assumption over the size of classes was re-established through the classless Inter-domain routing (CIDR), in order to allow better utilisation of the IP space. Within 15 years though the initial assumption over the size of the address space prooved also shortviewed, as IP addresses were not sufficient to cover the needs of hosts. A number of layer violations, like NATs, were widely used within the subsequent years in order to provide connectivity to the increasing number of end-hosts. In order to address this problem within the design of the network protocol, a revised version of IP has been proposed Deering and Hinden [1998] since 1998, but its deployment is slow, as the size of the current Internet makes it extremely difficult to replace IPv4 without significant connectivity problems and costs.

Resource allocation : Internet provides a best-effort forwarding mechanism. This design decision was chosen in order to enforce the end-to-end principle of the Internet Saltzer et al. [1984] and avoid state in the intermediate nodes of the network. Such an approach covered sufficiently the requirements of the networked applications of the time. As new network application became available over the years, more strict performance requirements were introduced. Unfortunately, Internet currently has no mechanism to address these requirement network-wide. Network engineers have tackled this problem through adequate resource provision Teitelbaum and Shalunov [2002]. This approach though becomes inefficient as network rates increase. In a 40gbps link the impact of queueing delays or packet drop becomes significant to the performance of streams. In related literature, a number of approaches has been proposed to address this problem in multiple layers of the network stack Blake et al. [1998]; BORDER et al. [2001]; Kuzmanovic et al. [2009]. Unfortunately, such approaches are difficult to deploy across large networks, as they require significant upgrade in network elements, introducing a significant cost.

Bidirectional connectivity : A side-effect of mechanisms addressing the previous two problem is the collapse of a fundamental assumption of computer network design, the ability of two connected nodes to communicate. A node which is behind a traffic inspecting middlebox is not guaranteed to receive incoming connections from any node and thus is not fully interactive. This problem has a direct consequence for users to resolve to 3rd party services in order to establish connectivity, changing as a

result the communication mechanism.

A number of problems that we experience with current network functionality can be traced back to the assumptions of the protocols. A number of clean slate approach have been proposed over the years, that address a number of these problems. The process though to deploy a new protocol is not straightforward. Computer networks currently suffer from an effect that is term as {it 'protocol ossification'} in the research community. The protocol hierarchy in the internet currently looks like an hourglass. A multitude of protocol exist in the application and link layer, but we only have IP in the network layer and TCP and UDP in the transport layer. The specifications of these protocols defined a number of mechanisms that allow protocol designers to develop extensions. Unfortunately, these mechanisms are not guaranteed to be supported across the network, as it is not critical for functionality and thus can be sacrifices in favour of performance. As a result, the capabilities to evolve protocol in a manner that is compatible with the current Internet infrastructure is impossible. In Bauer et al. [2011] authors report that 80% of popular services doesn't support ECN and 0,6% of destination may drop ECN traffic, while in Honda et al. [2011] authors report a large scale inability of the Internet to cope with TCP traffic that carries unknown option fields.

1.2 Contributions

1.3 Outline

1.4 Publications

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

- Rotsos, C., Van Gael, J., Moore, A. W., & Ghahramani, Z. (2010). Probabilistic graphical models for semi-supervised traffic classification (pp. 752757). Presented at the IWCMC '10: Proceedings of the 6th International Wireless Communications and Mobile Computing Conference.

- Mortier, R., Ben Bedwell, Glover, K., Lodge, T., Rodden, T., Rotsos, C., et al. (2011). Supporting novel home network management interfaces with open-

flow and NOX. Presented at the SIGCOMM '11: Proceedings of the ACM SIGCOMM 2011 conference, ACM. doi:10.1145/2018436.2018523

- Madhavapeddy, A., Mortier, R., Gazagnaire, T., Proust, R., Scott, D., Singh, B., et al. (2011). Constructing a Functional Cloud (Mirage 2011), 110.

- Mortier, R., Rodden, T., Lodge, T., McAuley, D., Rotsos, C., Moore, A. W., et al. (2012). Control and understanding: Owning your home network (pp. 110). doi:10.1109/COMSNETS.2012.6151322

- Rotsos, C., Sarrar, N., Uhlig, S., Sherwood, R., & Moore, A. (2012). Oflops: An open framework for openflow switch evaluation, 8595.

- Chaudhry, A., Madhavapeddy, A., Rotsos, C., Mortier, R., Aucinas, A., Crowcroft, J., et al. (2012). 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, ACM.

- Rotsos, C., Mortier, R., Madhavapeddy, A., Singh, B., & Moore, A. W. C. I. 2. I. I. C. O. (n.d.). Cost, performance & flexibility in OpenFlow: Pick three. Presented at the Communications (ICC), 2012 IEEE International Conference on.

- Madhavapeddy, A., Mortier, R., Rotsos, C., Scott, D., Singh, B., Gazagnaire, T., et al. (2013). Unikernels: Library Operating Systems for the Cloud. Proceedings of ASPLOS.

Chapter 2

Background

2.1 Packet forwarding

2.1.1 Data link layer

2.1.2 Network layer

2.1.3 Transport layer

2.2 Forwarding Control

2.2.1 Routing and switching

2.2.2 Switchlets and Active Networks

2.2.3 SDN

2.3 Control Plane Applications

2.3.1 Datacenter network

2.3.2 Home network

2.3.3 Wireless network

2.3.4 Simulation

2.4 Conclusions

Chapter 3

SDN control mechanism evaluation

In this chapter we present an extensive performance analysis 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 that enables SDN developers to study the performance of their network designs. The work focuses oni 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 develop a set of test scenarios that benchmark the performance of elementary OpenFlow protocol interactions. On the other hand, SDNSIM is a macro-benchmark OpenFlow platform, which extends the Unikernel abstraction and provides support for 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.

A bit strange. need to rephrase 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 different control architectures over the fat-tree topology (Section 3.8) and conclude (Section 4.6).

3.1 Network Control Micro-Characterisation

Despite the recent introduction of the SDN paradigm, the research community has already proposed a wide range of applications that take advantages of the protocol capabilities Handigol et al. [2009]; Sherwood et al. [2010]; Yu et al. [2010]. 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 is critical. Modifying established network control mechanisms must ensure these requirements and requires extensive testing and performance characterisation. An example of an early OpenFlow deployment that faced significant problems in a production environment is reported in Weissmann and Seetharaman. 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, as the deployed hardware switching platform was unable to handle the rate of the control channel. A significant reason of this problem can be traced back to the fact that in the SDN application ecosystem, there is a luck of an established and global mechanism to assess switch performance. In existing switching devices, performance is defined through the speed and capacity of the switching fabric as well as the size of the mac cache. The OpenFlow protocol increases significantly the degrees of freedom in user - device interaction and the characterisation task is complex.

In order to address this issue we developed OFLOPS [1], a measurement framework that enables rapid development of performance tests for both 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].

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

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

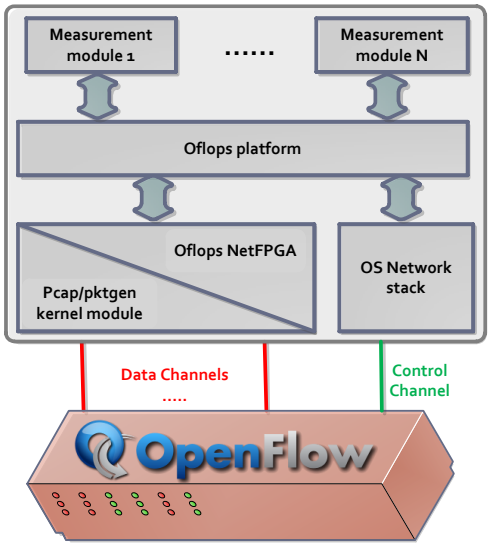


Figure 3.1: OFLOPS design schematic

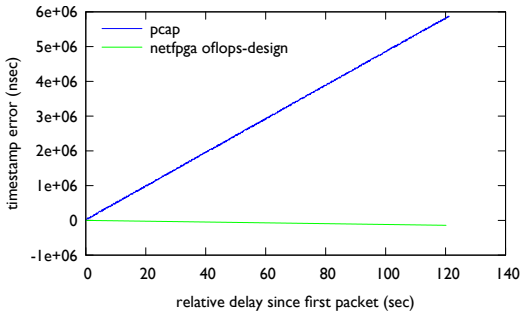


Figure 3.2: Evaluating timestamping precision using a DAG card.

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. Further, current controller designs, like SNA [2010]; Gude et al. [2008a], target production networks and thus are optimized for throughput maximization and programmability, but incur high measurement inaccuracy. Measurement noise suppression in the control plane requires a new simplified OpenFlow controller library with low processing latency. Finally, high precision measurements after a point are subject to loss due to unobserved parameters of the measurement host, such as 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 a rich, event-driven, API that allows developers 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 3.1. Details of the OFLOPS programming model can be found in the API manual ofl.

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, kernel-space through the pktgen module Olsson [2005b], and hardware-accelerated through an extension of the design of the NetFPGA Stanford Packet Generator Covington et al. [2009]. 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.

A comparison of the precision of the traffic capturing mechanisms is presented in Figure 3.2. In this experiment 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 the figure, 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

The number of OpenFlow-enabled devices has slowly increased recently, with switch and router vendors providing experimental OpenFlow support such as prototype and evaluation firmware. At the end of 2009, the OpenFlow protocol specification was released in its first stable version 1.0 ope [2009], the first recommended version implemented by vendors for production systems. Consequently, vendors did proceed on maturing their prototype implementations, offering production-ready OpenFlow-enabled switches today. Using OFLOPS, we evaluate OpenFlow-enabled switches from three different switch vendors. Vendor 1 has production-ready OpenFlow support, whereas vendors 2 and 3 at this point only provide experimental OpenFlow support. The set of selected switches provides a representative but not exhaustive sample of available OpenFlow-enabled top-of-rack-style switching hardware. Details regarding the CPU and the size of the flow table of the switches are provided in Table 3.1.

OpenFlow is not limited to hardware. The OpenFlow protocol reference is the software switch, OpenVSwitch Pettit et al. [2010], an important implementation for production environments. Firstly, OpenVSwitch provides a replacement for the poor-performing Linux bridge Bianco et al. [2010], a crucial functionality for virtualised operating systems. Secondly, several hardware switch vendors use OpenVSwitch as the basis for the development of their own OpenFlow-enabled firmware. OpenVSwitch development team has standardised a clean abstraction over the control of the switch silicon (similar to linux HAL), which allows code reuse over any forwarding entity that implements the switch abstraction. Thus, the mature software implementation of the OpenFlow protocol is ported to commercial hardware, making certain implementation bugs less likely to (re)appear. In this paper, we study OpenVSwitch 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 OpenFlow reference implementation, 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
OpenVSwitch	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 independent of the probe rate.

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 not different in this regard. The current reference implementation is defined through OpenVSwitch Pettit et al. [2010]. However, different software and hardware implementations may not implement all features defined in the OpenVSwitch 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 **??**, (2) the packet interception and packet injection functionality of the protocol **??**, (3) the update rate of the OpenFlow flow table along with its impact on the data plane, **??** (4) the monitoring capabilities provided by OpenFlow, and (5) the impact of interactions between different OpenFlow operations.

3.4.1 Packet modifications

The OpenFlow specification ope [2009] defines ten packet modification actions which can be applied on incoming packets. Available actions include modification of MAC, IP, and VLAN values, as well as transport-layer fields. 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. On the other hand, how these operations are to be used is yet to be defined for new network primitives and applications, such as network virtualization, mobility support, or flow-based traffic engineering.

To measure the time taken by an OpenFlow implementation to modify a packet field header, we generate from the NetFPGA card UDP packets of 100 bytes at a constant rate of 100Mbps (approx. 125 Kpps). This rate is high enough to give statistically significant results in a short period of time, without causing any packet queuing for any of the switches. The flow table is initialized with a flow that applies a specific action on all probe packets and the processing delay is calculated using the transmission and receipt timestamps, provided by the NetFPGA.

Evaluating individual packet field modification, Table 3.2 reports the median difference between the generation and capture timestamp of the measurement probe along with its standard deviation and percent of lost packets.

We observe significant differences in the performance of the hardware switches due in part to the way each handles 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 providing 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 OpenVSwitch software implementation lies between Switch1 and the other hardware

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

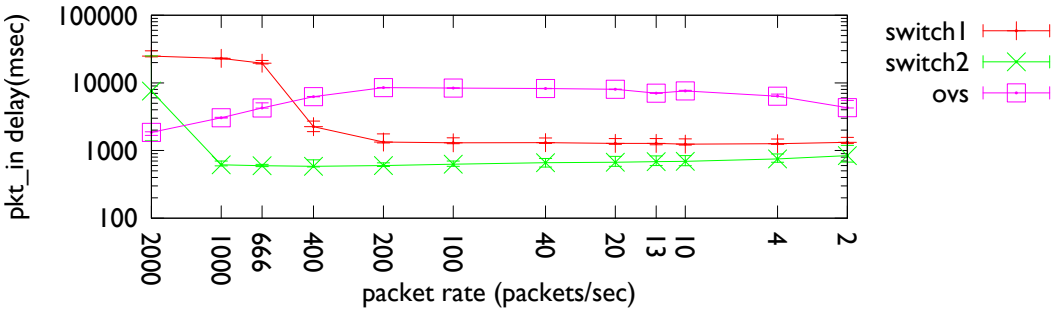
Table 3.2: Time in μs to perform individual packet modifications and packet loss. Processing delay indicates whether the operation is implemented in hardware ($<10\mu$s) or performed by the CPU ($>10\mu$s).

switches. OpenVSwitch fully implements all OpenFlow actions. However, hardware switches outperform OpenVSwitch when the flow actions are supported in hardware.

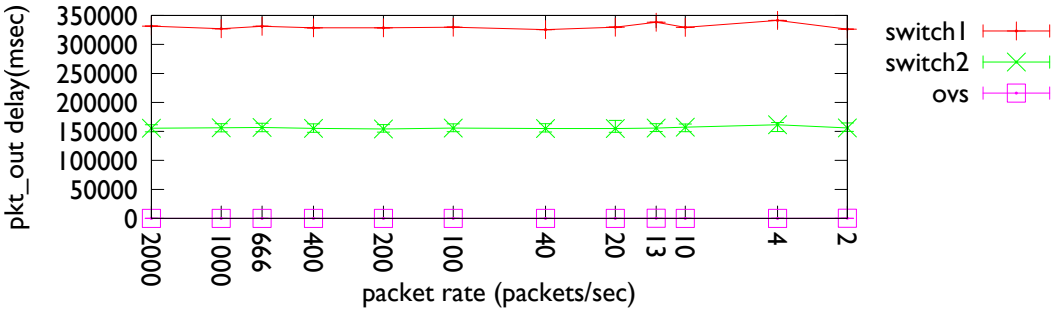
We conducted a further series of experiments with variable numbers of packet modifications as flow actions. We observed, that the combined processing time of a set of packet modifications is equal to the highest processing time across all individual actions in the set. Furthermore, we notice that for Switch1 and OpenVSwitch there is a limit of 7 actions, which potentially exposes some relation in the code base.

3.4.2 Traffic interception and injection

OpenFlow protocol permits a controller to intercept or inject traffic over the control plane. This functionality permits to OpenFlow controller applications to be reactive



(a) Packet in message latency



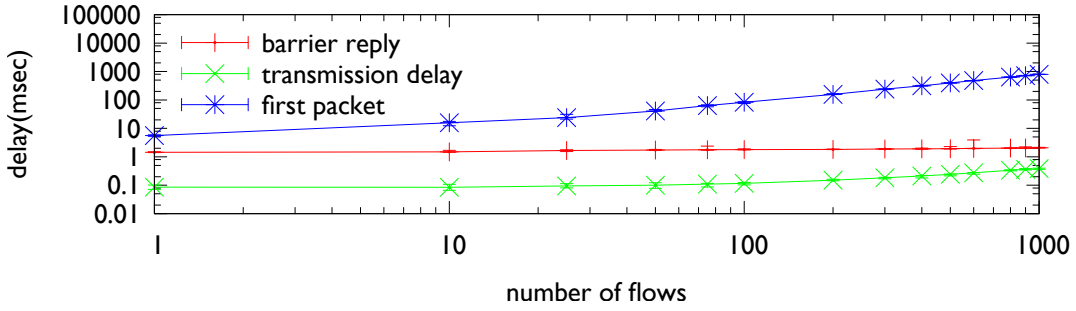
(b) Packet out message latenct

Figure 3.3: Latency to intercept or inject a packet using the OpenFlow protocol

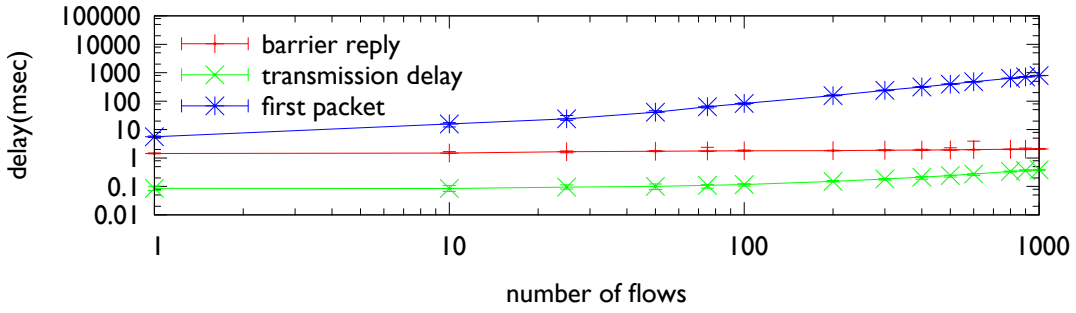
and handle traffic on a per-flow basis. Packet injection allows the controller to interact with connected network hosts. The interception mechanism in OpenFlow has been reported in the initials deployments of the protocol to cause significant slow-down in the control plane and led to switch disconnection at high packet rate Kobayashi et al.. This is a direct consequence of the silicon design in current OpenFlow switches, that develop such functionality over a low-frequency exception notification channel. In order to characterise this functionality, we develop a simple experiment using OFLOPS that sends packet_in messages over the control channel at different rates and measure the latency of the switch to process them. In Figure 3.3, we plot the latency induced on packets both for Packet_in and Packet_out messages. We omit in this experiment Switch 3 as this functionality cause high CPU utilisation and after a few seconds made the switch unresponsive. For packet_out messages, the switches rate limit through the TCP rate control mechanism the rate of messages received and as a result they provide

a constant performance. For packet_in messages, we observe a diverse behaviour between hardware switches at high packet rates. For Switch 1, packet loss and latency gets high for traffic rates above 400 packets per second. Additionally, we noticed that the switch is able to process a maximum of 500 packets/sec. For Switch 2 latency and packet loss are significantly lower and stable. Switch 2 faced problem to process packets at high rates, over 2000 packets per second. OpenVSwitch, has a high but stable latency for any tested data rates.

3.4.3 Flow table update rate



(a) OpenVSwitch (log-log scale)



(b) Switch1 (log-log scale)

Figure 3.4: Flow entry insertion delay: as reported using the `barrier` notification and as observed at the data plane.

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 ope [2009], 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 OpenVSwitch and Switch1, Figure 3.4 shows the time to install a set of entries in the flow table. The NetFPGA-based switch results (not reported) are similar to those of Switch1, while 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 10Mbps that targets the newly installed flows in a round-robin manner. The probe achieves sufficiently low inter-packet periods in order to measure accurately the flow insertion time.

In Figure 3.4, 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 3.4 shows the median result as well as the 10^{th} and 90^{th} percentiles (variations are small and cannot be easily viewed).

From Figure 3.4, 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

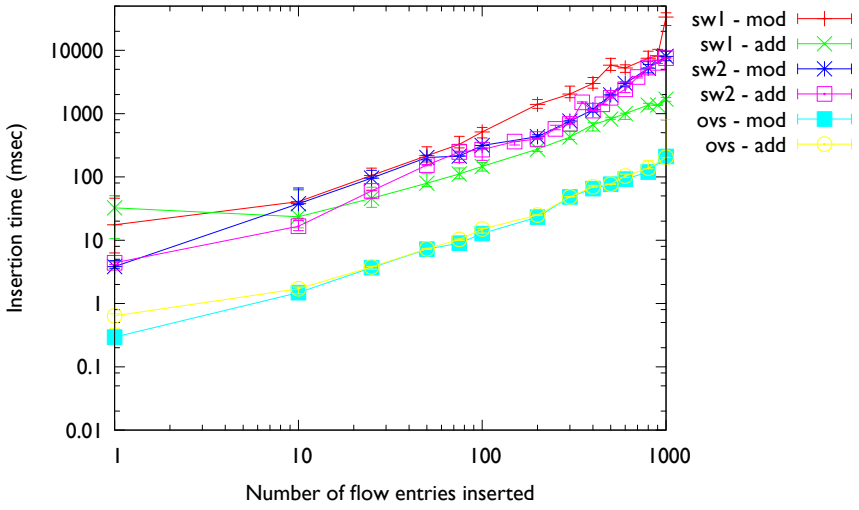


Figure 3.5: Delay of flow insertion and flow modification, as observed from the data plane (log-log scale).

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 3.5 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. Sur-

prisingly, 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.

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 TCAM. With such possible choices and range of different experiments, the flow insertion times reported in Figure 3.5 are not generalizable, but rather depend on the type of insertion entry and implementation.

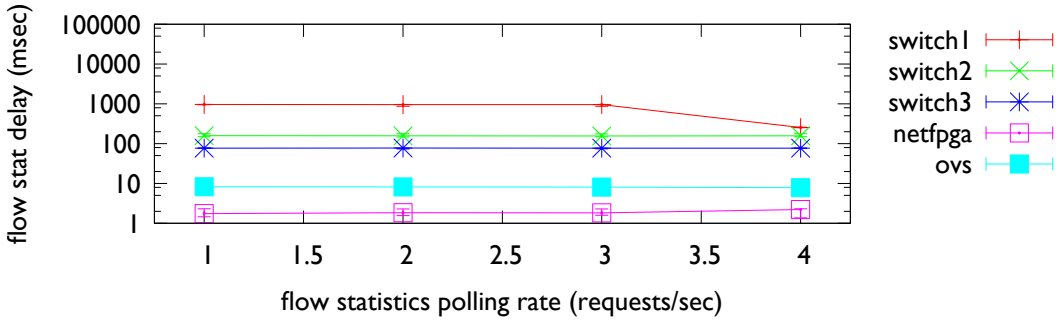
3.4.4 Flow monitoring

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.

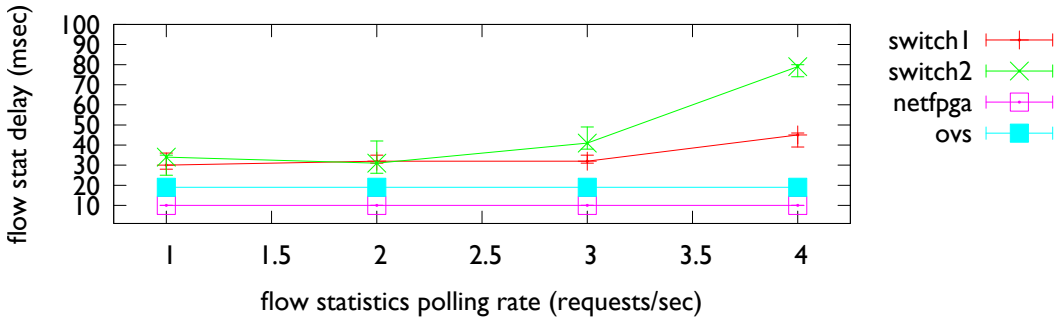
We now test the performance implications of the traffic statistics reporting mechanism of OpenFlow. Using OFLOPS, we install flow entries that match packets sent on the data path. Simultaneously, we start sending flow statistics requests to the switch. Throughout the experiment we record: the delay getting 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 3.6 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 results for even a few queries per second being sent. Figure 3.6 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



(a) Reply time.



(b) CPU utilization.

Figure 3.6: Time to receive a flow statistic (median) and corresponding CPU utilization.

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 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

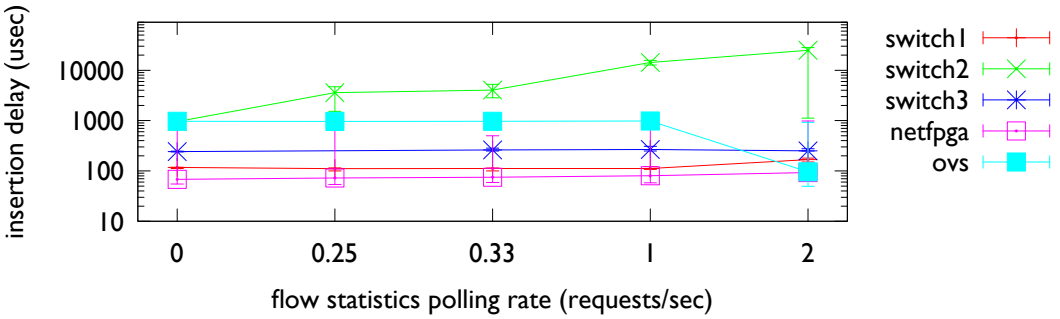


Figure 3.7: Delay when updating flow table while the controller polls for statistics.

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 applications with, e.g., flow arrival notifications from the network, while simultaneously providing fine-grain control of the forwarding process. This permits applications to adapt in real time to the requirements and load of the network Handigol et al. [2009]; Yap et al. [2009]. Under certain OpenFlow usage scenarios, e.g., the simultaneous querying of traffic statistics and modification of the flow table, understanding the behavior of the data and control plane of OpenFlow switches is difficult without advanced measurement instrumentation such as the one provided by OFLOPS. Through this scenario, we extend Section 3.4.3 to show how the mechanisms of traffic statistics extraction and table manipulation may 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 for the modified flows to 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 3.7 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 explained by the CPU utilization increase incurred by the flow statistics polling (Figure 3.6(b)). Finally, OpenVSwitch 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 cbe, provide a sufficient set of tools to profile functionality of OpenFlow building blocks and understand the low-level capabilities. Nonetheless, the provided measurements are not sufficient to establish models that can predict network performance with tight error bounds. The distribution of control functionality over multiple functional units, in conjunction with the diverse behaviour of OpenFlow switch and controlling platforms, reduce the ability to develop analytical models that can estimate the behaviour of an SDN design. In order to reason on performance, as well as correctness, developers have to revert to an experimental approach. In the related literature on network evaluation there have been two main experimental mechanisms: *realistic-testbed* and *simulation*.

Realistic testbeds try to reconstruct in full detail the properties of the deployment environment. This approach provides an optimal measurement environment with complete control over the parameter of the experiment, but has a significant overhead in terms of resource requirements and configuration time, which scales badly as the experiment size increases. Setting up a realistic testbed for datacenter networking requires a large number of machines and network devices with identical functionality in respect to the deployment environment, interconnection planning and careful metrication and analysis of the resulting system. In an effort to improve the scalability issues of realistic testbeds, the research community has established a number of shared testbeds. Such testbeds employ techniques such as virtualization and statistical multiplexing, and provide low-level user access to sizable infrastructures emu; pla. Non-

theless, such platform are not always a good fit for network experiments. Resource control is reduced and measurement noise, due to infrastructure sharing, is not always easy to detect and remove.

In the simulation approach, researchers replace parts of the functionality of the system by simpler models Varga and Hornig [2008]; **?**. Such approaches aim to reduce the complexity of an experiment for large scale networks, but faces a number of limitation. Firstly, the fidelity 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 in order to fit the abstraction of the underlying models. For example, POSIX socket-based controllers need to modify the control channel abstraction in order to match the API of the simulation platform, while forwarding plane traffic may have to be translated in a stochastic model.

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 implement the desired network functionality over the mirage OS abstraction, and at the compilation step produce a number of different experimentation target. SDNSIM provides two experimentation options: *Simulation*, transforms the high level logic in NS-3 **?** simulation, and *Emulation*, translates the high level logic of each host into Xen-based interconnected DomU VMs. In addition, because SDNSIM reuse the abstraction provided by the Mirage OS, functionality can also be translated to any of the available output of the Mirage OS.

3.6 Mirage Library OS

Mirage is a cloud service development framework written in OCaml. Mirage applications are single purpose appliances that are compile-time specialised into standalone kernels deployable on the Xen platform. The aim of the framework is to provide small size cloud OS images that are efficient and secure. In order to achieve this, Mirage revisits the idea of library OS; OS functionality is separated into logical modules and

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

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

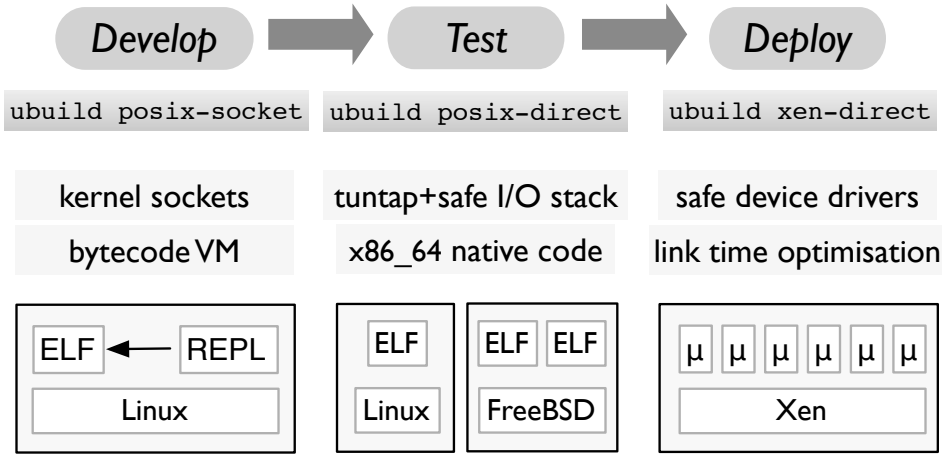


Figure 3.8: Specialising a Mirage application through recompilation alone, from interactive UNIX Read-Eval-Print Loop, to reduce dependency on the host kernel, and finally a unikernel VM.

added to an appliance only if the code expresses an explicit dependency. As a result, Mirage can generate small size VM images with very fast boot times. Furthermore, using OCaml, a type-safe functional language, the framework is able to mitigate a number of security attacks to applications.

Mirage executes OCaml code using a specialised language runtime modified in two key areas: *memory management* and *concurrency*. Since Mirage applications are single process VMs, traditional complex memory virtualisation and Address Space Randomisation (ASR) mechanisms are removed from the architecture. Mirage applications use a single address space, separated between the text and data section of the program and the runtime heap. In addition, since the program code is immutable during runtime, Mirage locks write access to executable memory space, thus mitigating buffer overflow attacks. Finally, in order to improve performance for our system, Mirage provides a memory-safe zero-copy mechanism and exposes applications to the memory space of the shared memory ring.

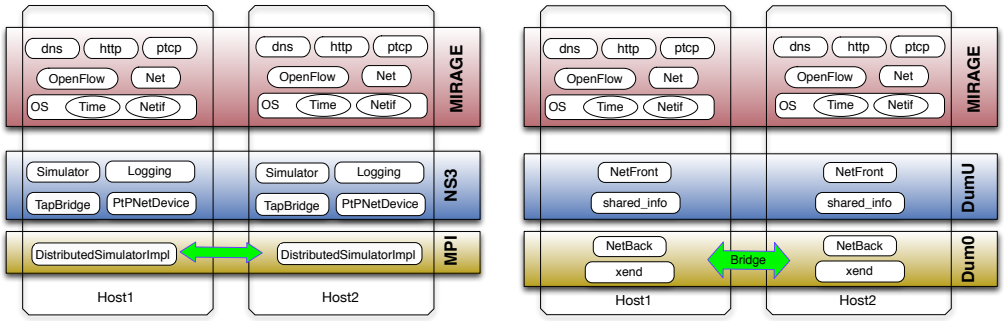
In terms of concurrency, Mirage uses the Lwt cooperative threading library abstraction lwt. Lwt provides an OCaml syntax extension that can annotate blocking IO and internally evaluate blocking functions into event descriptors to provide straight-line control ow for the developer. Written in pure OCaml, Lwt threads are heap-allocated

values, with only the thread main loop requiring a C binding to poll for external events. Mirage provides an evaluator that uses Xen polling to listen for events and wake up lightweight threads. The VM is thus either executing OCaml code or blocked, with no internal preemption or asynchronous interrupts. The main thread repeatedly executes until it completes or throws an exception, and the domain subsequently shuts down with the VM exit code matching the thread return value. A useful consequence is that most scheduling and thread logic is contained in an application library, and can thus be modified by the developer as they see fit.

Mirage OS provides a simple API to applications developers, sufficient for systems programming. This functionality is implemented by two core modules, named *Net* and *OS*, which expose a minimum API to the network and the device management stack. The simplicity of the OS and Net modules, permit Mirage to compile code to other target backends, apart from the Xen platform. Specifically, Mirage can generate UNIX binaries, using both the POSIX library network functionality and raw sockets, and even Javascript executables that run in a browser. There is also currently an effort to port Mirage in the FreeBSD kernel as well as over the BareMetalOS bar, an assembly OS. The diverse set of deployment backends, provides a sufficient environment for test and optimization, as depicted in Figure 3.8. Developers build initially their core logic over the POSIX backend in order to test the correctness of the code, then they can try their code over the Mirage default network stack, to perform a small scale performance evaluation, and finally they can synthesize the resulting deployable Xen Image.

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>
```



(a) NS3 (b) Xen

Figure 3.9: SDNSIM host internal architecture: NS3 simulation 3.9(a) and xen real-time emulation 3.9(b).

```
    <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"/>
</topology>
```

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

From the user 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. For an experiment definition, developers needs to define, at minimum, 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. For each host users must define the host name (node@name) and the host main function (node@main), while a number of named parameters (node/-

param) can be passed to the main function. Finally, developers can define links between 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. Links can be used also to integrate external network interfaces in the simulation, in order to allow the experiment to interact with entities outside of the experiment scope.

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. At the top layer of the host is the application logic of the host. This layer is programmed by the developer, and define the traffic load of the network and its forwarding logic. In order to allow realistic traffic patterns, SDNSIM can reuse all the application protocol libraries supported in the Mirage platform, namely DNS, HTTP, SSH and OpenFlow. For the OpenFlow protocol library we modify the controller and switch functionality and expose control hooks to define explicit lower bounds in processing latencies for control and forward plane interactions. This allows developers to inject measurement results from OFLOPS and Cbench in their experiments. Additionally, we have reimplemented in OCaml the pttcp tcp test tool **??**, in order to allow model driven TCP and UDP traffic generation.

In the middle layer of the host architecture, we reuse the network library and the OS abstraction defined by the Mirage platform. These libraries are mapped in the lower layer of the respective backend. Because these platforms aim to develop an OpenFlow capable simulation platform, the main focus for the integration between the two lower layer is the fidelity in network functionality and time consistency. Currently, SDNSIM supports two backends, *NS3* and *Xen*. In order to implement the lower layer integration, a different strategy has been followed for each backend.

3.7.1 Xen

Mirage Xen Image, use a simple PV Boot mechanism. The mechanism initializes a VM with a single virtual CPU, loads the Xen event channel and jumps to the main function of the framework. The main function enumerates IO devices and notifies the application and then progresses to the thread scheduler. As we have mentioned earlier, the threading mechanism in Mirage is based on Lwt, a language extension which enables seamless event-driven asynchronous programming. Using Lwt suntax, each closure can be noted as blocking, and spawn a new lwt thread. At the lowest level, an

lwt thread is either tied to a blocking IO request, or a time dependent event. Using this information, the scheduling logic works as follow: If no sleeping thread is currently available to resume, the schduler calculates the time left until the next time event and uses it as the timeout parameter for the IO blocking method (named *d*omainpoll) provided by the Xen platform. Domainpoll registers interest to the respective event handler and ask the Xen scheduler to put the VM to sleep until either an event occurs or the timeout option expires.Timing integration with the Xen platform is achieved through the Xen shared_info struct, a structure shared between the Dum0 and the DumU VMs. Network functionality is implemented over the IO interface of the Xen NetFront driver.

SDNSIM uses Xen Managment API (XenApi)to control the experimental configuration over the Xen platform. XenApi provides remote VM and resource allocation control of a Xen Domain. As a result, SDNSIM is able to create and start VM instances, implements network topologies through vif bridging in the Dum0 space, and assign link rate and propagation delay to vif devices. In a Xen-based emulation, SDNSIM will firstly compiles all required VM images, creates the appropriate host definition and network topologies over the Xen platform and, in the end, start all VM images.

3.7.2 NS3

NS3 is a discrete time packet driven network simulation framework. The core of the system consists of a discrete time event engine, while a set of NS3 libraries provide an extensive set of network applications, routing protocols, data-link layer protocols and link emulations models. NS3 is widely used in academia and is considered as the default simulation tool in the domain of MANETs and wireless communications.

The NS3 programming abstraction has a significant difference from available Mirage backends. The event engine of NS3 is blocking, and thus a bad match for the default Lwt thread scheduler. When an event occurs in NS3, the event engine propagates the event to the registered event handler and the execution is blocked until the event handler returns. The default Lwt thread scheduler requires an infinite while loop in the main function of the program, in order to implement its cooperative scheduling. In order to implement the Mirage abstraction over the NS3 engine, we had to integrate the thread scheduling engine with the event engine. In terms of the time abstraction,

the OS clock is bridge with the NS3 simulation clock, while each sleep call is blocked and scheduled as an NS3 time event. The thread is resumed from a sleep call when the simulator fires the respective time event. IO scheduling is integrates with the network device abstraction of NS3. We register an event handler in the NS3 event engine which can push data to the network thread and reschedule it, when a packet is received. Finally, in order to avoid scheduling deadlocks the OS schedules *i*dle time events that resume any yielded threads.

Network connectivity uses the link abstraction of a *P*ointToPoint channel. This model simulates a PPP link over a lossless medium, a valid approximations for the full duplex non-shared medium of current network datacenters. Traffic transmission, uses a single packet queue per network device shared between the Mirage layer and the NS3 simulator engine. We modify the default NS3 device functionality and provide a channel to report back pressure from the queue to the network stack of Mirage.

A performance limitation that we faced during the development of the NS3 backend for SDNSIM is the natural inability of the OCaml runtime to support multi-core programming. This is a core design decision for OCaml, which provides predictable performance and avoids garbage collector synchronisation delays. In order to make SDNSIM scalable for large network sizes we employed a distributed version of the NS3 event engine which uses MPI for inter-process communication Pelkey and Riley [2011]. This simulation mechanism uses a simple and conservative clock synchronisation mechanism, that ensures that all events are executed in order.

3.8 SDNSIM evaluation

In order to evaluate the performance of the SDNSIM platform we develop a number of small scale micro-benchmarks that evaluate 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 Section **??**, we use two off-the-self OpenFlow benchmarking platforms in order to characterise the performance of the controller and switch implementation. Further, in Section 3.8.3 we characterise the scalability of the NS3 backend.

3.8.1 Mirage Controller

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 publicly available 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, a highly optimised version that sacrifices Python integration for better performance. Maestro Cai et al. [2011] is an optimised Java-based controller that aims to achieve fairness among switches. We compare these against the Mirage controller targeting two different network backends: *mirage-unix* targets the UNIX Sockets backend and so uses the existing Linux TCP/IP stack, while *mirage-xen* targets the Xen hypervisor and runs as a domU virtual machine using the Mirage TCP/IP stack.

Our benchmark setup uses the *cbench* application[1]. Each emulated switch simultaneously generates *packet-in* messages and the program measures the throughput of the controller in processing these requests. 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 switch maintains a full 64 kB buffer of outgoing packet-in messages. The first measures the throughput of the controller when serving connected switches fairly, while the second measures absolute throughput when servicing requests.

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 Mirage does not yet support multi-threading. For each controller we run the experiment for 120 seconds and measure the per-second rate of successful interactions. Table 3.3 reports the average and standard deviation of requests serviced per second.

Unsurprisingly, due to mature, highly optimised code, *NOX fast* shows the highest performance for both experiments. However, we note that the controller exhibits extreme short-term unfairness in the throughput test. *NOX* provides greater fairness in the throughput test, at the cost of significantly reduced performance. Maestro performs

[1] http://www.openflow.org/wk/index.php/Oflops

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 UNIX	68.1	11.7	21.1	0.2
Mirage Xen	86.5	4.4	20.5	0.0

Table 3.3: OpenFlow controller performance.

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 with both backends; the Xen backend wins out over the UNIX backend due to reduction of layers in the network stack. In addition, Mirage Xen achieves the best product of performance and fairness among all tested controllers in the throughput test. Comparing latency, both Mirage backends perform much better than Maestro but suffer somewhat in comparison to NOX: we believe this is due to the lack of optimisation in the Mirage TCP/IP stack.
Add latest results

3.8.2 Mirage Switch

We also use the OFLOPS benchmark platform **?** to evaluate performance of the Mirage switch implementation. We compare against the Open vSwitch[1] (OVS) kernel implementation, an OpenFlow-enabled software switch implemented as a Linux kernel module. OVS is currently used by many datacenter service providers to enable virtual machines to be bridged in dom0, while its OpenFlow functionality is used by vendors to implement OpenFlow firmware.

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. This represents a setup that might be used to enable an application to modify switch functionality without affecting the network functionality in dom0. Using Oflops, we generate packets on one of the data channels

[1] http://openvswitch.org

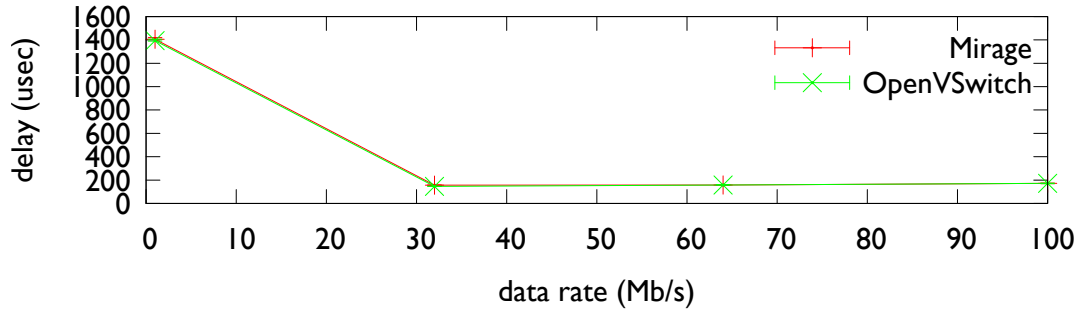


Figure 3.10: Min/max/median delay switching 100 byte packets when running the Mirage switch and Open vSwitch kernel module as domU virtual machines.

and receive traffic on the other, having inserted appropriate flow table entries at the beginning of the test. We run the test for 30 seconds using small packets (100 bytes) and varying 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 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 due to space constraints, but can report that both implementations suffer similar levels of packet loss. However, the Mirage switch has a memory footprint of just 32 MB compared with the Open vSwitch virtual machine requirement of at least 128 MB. We are currently working toward better integration of the Mirage switch functionality with the Xen network stack to achieve lower switching latency. As a result

3.8.3 NS-3 performance

In order to test the scaling properties of the NS3 backend we perform a simple topology experiment, depicted in Figure (Figure 3.11). The topology consists of a number of switches and an even number of hosts, splitted in pairs and generating steady state TCP between each pair. We use two variations of the topology: A centralised topology where all hosts are connected to a single switch, and a localised topology, where hosts are distributed between two switches and traffic remains local to the switch. Each switch is connected to an OpenFlow controller that implements a learning switch. The

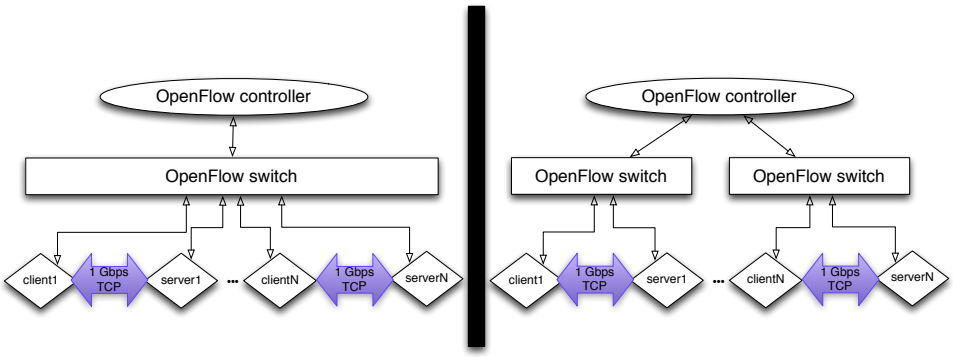


Figure 3.11: Topology of two basic simulation scenarios for the SDNsim platform

	Single Switch			two switches		
Number of hosts	2	8	12	4	8	12
Delay (in min)	17	90	171	21	50	82
Slowdown factor	34	180	242	42	100	164

Table 3.4: SDNSIM simulation of a fat-tree topology over NS3 backend allows better scaling of the slowdown factor as the traffic is localised.

experiment executes 30 seconds of simulation time.

In Table 3.8.3, we present the real execution time and the slowdown factor of each simulation. The results show that the platform can scale close to linear when the hosts of the simulation create small autonomous partition. In the centralised topology, the OpenFlow switch becomes a bottleneck of the simulation, since it has to process sequentially all network events. In the localised topology, the distributed nature of the event engine permits parallelization of the event processing between the two switches, thus reducing the experiment running time.

3.9 Security Tradeoffs on Datacenter Network Micro-control

Maybe remove this section

3.10 Summary and Conclusions

Add some notes on SDNSIM 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.

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.

We hope that the use of OFLOPS will trigger improvements in the OpenFlow protocol as well as its implementations by various vendors.

Chapter 4

Home network control scalability

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

In Section 4.1 we present a thorough review of ethnographic and social studies and elaborate on the nature of the problems and the inherent opportunities of the specific network environment. In Section 4.3, we describe our home router and how its flow-based approach enables it to help improve the user experience. In section 4.4 we present and evaluate protocol modifications that place the homeowner in more direct control of their network. In section 4.5 we present a simple QoS policy mechanism that provides a communication channel between the home owner and the ISP and enables a user friendly traffic scheduling mechanism for the ISP build using commodity SDN applications. Finally, in Section **??** we conclude the results of our exploration.

Note that throughout this Chapter we refer to the individual managing the home network as the homeowner without loss of generality; clearly any suitably permitted 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

Consumer broadband Internet access is a critical component of the digital revolution in domestic settings: for example, Finland has made broadband access a legal right for all its citizens [1]. A growing number of services are now provided over the Internet, including government, entertainment, communications, retail and health. The growth of IP enabled devices over the last decade also means many households are now exploring the use of in-home wired and wireless networking, not only to allow multiple computers to share an Internet connection but also to enable local media sharing, gaming, and other applications. In computer network literature, a number of studies have been published that highlight the distinct properties of the home network environment. In Subsection 4.1.1 we present briefly the connectivity and traffic mix characteristics of home networks based on the outcome of relevant studies, while in Subsection 4.1.2 we discuss the relation of home networking technologies with the social context of the home, based on 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. Internet connectivity is through ADSL or cable connection, while there are also other less popular links types found across the work, like fiber, 3g, satelite etc. 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.

Home network measurement studies have focused both on the local network properties as well as the nature of the Internet traffic. Local network analysis provides useful insight on the way devices interact within the house and the possible performance limitations. Such measurement studies focused mainly on developing active or passive measurement tools, that runs on end-hosts in the network and collect data. In Cioccio

[1] http://www.bbc.co.uk/news/10461048

et al. [2011] authors develop an end-host active measurement tool accompanied by a small user survey, named HomeNet Profiler. The analysis of the data unveils some interesting insights on home network topology and connectivity. In terms of the size of home networks, users report that their network consists on average of 7-8 device, but during the measurement on average only 1-2 device were active in the network. Further, the user survey reports a wide range of connected devices (e.g. smartphones, tablets, game consoles, etc.), which require from any designed solution to be as much as possible open to device heterogeneity. Finally, the study also reports useful statistics on wireless connectivity of end-hosts. The study reports that wifi connectivity to the home router is on average pretty good (signal strength ¡ -80dBm), but the medium appears to be over subscribed, with on average 10 active ssid discovered from the end-host wifi adapter during the measurement. Further approximately 1/3 of the wireless networks exhibit channel overlap with neighbouring networks. Interestingly, these observations point out the evolution of wireless technology and contrast earlier results on Yarvis et al. [2005]. In terms of traffic mix, in Reggani et al. [2012] authors analyse network traffic from a set of hosts in order to understand how users use network applications in different environments. In the home network environment, users tend to have distinct traffic mix. Specifically, network filesystem and P2P applications generate significant traffic volumes in the home environment, while a large portion of the traffic remains local and can only be observed from within the network.

Netalyzer Kreibich et al. [2010], a web based java-applet that tests network connectivity and protocol openness, provides useful insight on home networking. Analysis of collected data unveiled a significant number of protocol misconfiguration and malbehaving middleboxes in ISP networks. Additionally, using a simple buffer flooding technique, the authors detect large packet buffer in commercial home routers that can increase network latency up to 200 msec. Additionally, in Hätönen et al. [2010] authors test a number of off-the-self home router kit and discover that router firmwares exhibit a wide range of behaviours in terms of NAT functionality and DNSEC support. Nontheless, because of the popular usage of such routing kit, the impact to home networking appears to be minimal and home network environment appears to be resilient to non-standard router implementations.

In terms of Internet connectivity, a number of studies have tried to analyse residential networks in order to understand the properties of the traffic and the impact of the

ISP policies. Such analysis is also important at a governmental level, in order to asses the quality of the ISP product and the satisfiability of the clients FCC; **?**. Analysis of home network link properties relies to a great extend on BRAS level packet traces, or active measurement probes from specific monitoring points. One of the first analysis on this scope can be found in Dischinger et al. [2007a], where an active measurement was contacted in order to understand the properties of the link in residential networks. The results of the analysis highlighted the important performance differences between ISPs. The analysis pointed out that the bottleneck of the end-to-end path for such networks was detected on the last mile of the link, between the users modem and the BRAS of the ISP, while ISPs performance is highly variable over the day. In Sundaresan and de Donato [2011] authors follow a different approach in measuring the network link and integrate various measurement mechanisms in the home router firmware. In their analysis they unveil a number of interest differences in the performance of a number of hosts and point out that the very idea of performance is not clear and measurable by a single test, while critical performance factors are distributed in various points in the network.

In terms of home network Internet traffic mix, research has highlighted highly variable and location-dependent trends. In Cho et al. [2006] authors describe that during the time of their analysis in Japan there was a significant usage of P2P applications. More recent studies Maier et al. [2009] point out that users in Germany have shifted interest towards web applications and a large portion of the traffic in their trace was HTTP-based. Similar results are pointed out in Erman et al. [2011], where http traffic is analysed into application types, and online video and one-click hosting services appear as the predominant application classes.

4.1.2 Home Network as a social activity

Home networking technologies have been an interesting domain of study and application for HCI, Ubiquitous computing and sociology, since it provides an excellent environment to study the interaction between users and technology. Studies in the fields usually engage in user interviews in order to understand how users perceive and interact with technology.

An important aspect in this is understanding how people perceive home network

technologies. In Grinter and Edwards [2005]; Shehan-Poole et al. [2008] the authors ask from home network users to sketch their understanding of the home network. In the sketch analysis the authors highlight two important observations; user opacity to the network increases inversely proportional to their network experience – establishing the effectiveness of the deep abstraction-based design of the current network stack, and users characterize network devices within the context of the home network. Further, in Tolmie et al. [2007] the authors perform an empirical analysis of the house members with respect to technology and the home network. Interestingly, their finding detect that on average users are least motivated to interact with the home network in order to optimize it as long as the perceived performance is tolerable. Additionally, network maintenance is acceptable if it can resemble in format the other household duties (well defined and simple tasks with short durations), while the interest conflict that arise due to the shared nature of some infrastructure is usually solved through negotiation between the household members. An interesting study on this field is presented in Chetty et al. [2010]. In this study, the authors develop a visualization system that inform home network users with statistics on the network bandwidth usage. Interestingly, the introduction of such a mechanism made users informed on the way the network functions and how connectivity problems can be traced to network problem or to other users. Nontheless, such a technology also hinders the danger for users to expose personal informations.

A number of user studies have augmented the factors that shape home networking adding as an important factor the design of the building within which the network is installed. In Rodden and Benford [2003] authors describe a 7-layers model devised by the American writer Stewart Brand in Brand [1995] which describes how homes evolve architecturally after their initial establishment. Using this model authors analyse the relationships between Ubquitous technologies and home design. This study is further focused on the home networking technologies in Chetty et al. [2007]. Authors xontact a user study in order to understand how the home design relates to the choices of users regarding their home network. 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 Crabtree et al. [2003]; Rodden et al. [2004] the

authors study a number of family homes and monitor the real time communications and the ways in which information is produced and consumer within the house. In their study they concluide that a lot of these activities have a location reference within the house, which is related to the involved member as well as the house planning, while activities can be synthesized as sequences into higher order activities. Additionally, using these observations, they propose a framework which can model such interactions.

Finally, in Shehan and Edwards [2007] authors analyze some common management and configuration problem 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 developed for the Internet since 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 tech-

nology 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 company. Tension is created between them 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!

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;
- 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.

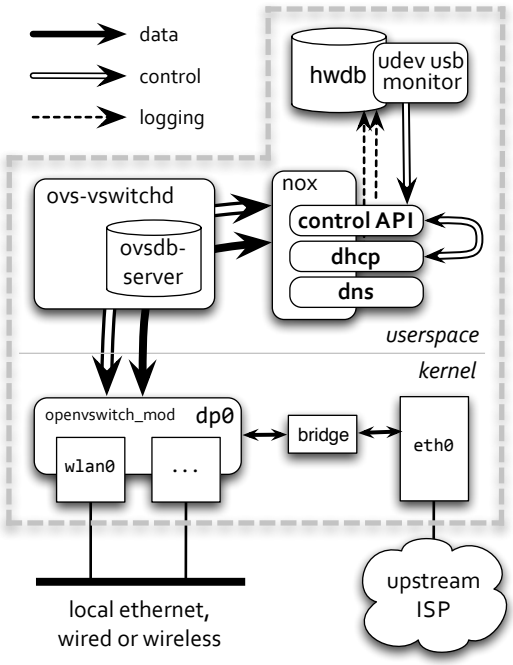


Figure 4.1: Home router architecture. Open vSwitch (*ovs**) and NOX manage the wireless interface. Three NOX modules provide a web services control API, a DHCP server with custom address allocation and lease management, and a DNS interceptor, all logging to the Homework Database (*hwdb*) (§4.4).

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 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.

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

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.

[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,[1] 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[2] controller as it provides a programmable platform abstracting OpenFlow interaction to events with associated callbacks, exporting APIs for C++ and Python.

Our functionality is implemented in 5 different Nox modules. C++ module *hwdb* synchronizes router state with the hwdb home database (Subsection 4.3.2), C++ module *homework_dhcp* implements our custom DHCP server described later in Section **??**, C++ module *homework_routing* implements the forwarding logic of the design, C++ module *homework_dns* implements the DNS interception functionality and 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 Open-

[1] http://openvswitch.org/
[2] http://noxrepo.org/

Flow datapath managing the wireless interface of the platform.[1] We provide NOX modules that implement a custom DHCP server, control forwarding, control wireless association via filtering, and intercept DNS lookups. Control of these modules 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]

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. [2005] 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 performance; and *Leases*, recording mappings assigned via DHCP.

[1] Without loss of generality, our home route 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 necessary to do so.

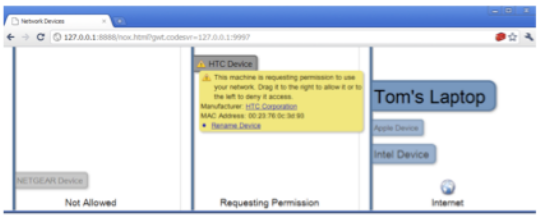


Figure 4.2: The *Guest Board* control panel, showing an HTC device requesting connectivity.

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 §**??**. 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. The interface is implemented in HTML/CSS/Javascript allowing it to be displayed on a range of devices, currently under trial with users. The router's measurement and control APIs described above are also being used to build a wide range of other interfaces for use via smartphones, web browsers, and custom display hardware.

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 negotiate with a server their connectivity parameters. As part of our design we extend the

functionality of the protocol to achieve two goals. First, we enable the homeowner to control which devices are permitted to connect to the home network by interjecting in the protocol exchange on a case-by-case basis. We achieve this by manipulating the lease expiry time, allocating only a short lease (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.

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.

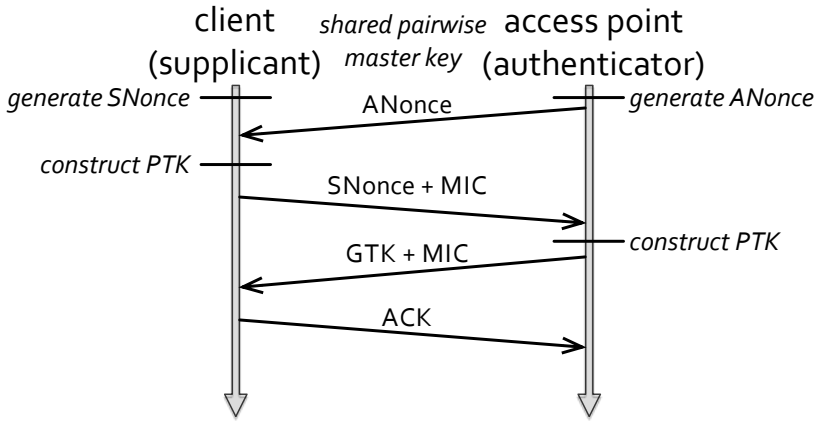


Figure 4.3: 802.11i handshake, part of the association process. Note that MIC (Message Integrity Code) is an alternate term for MAC, used in such contexts to avoid confusion with Media Access Control.

Our home router supports wireless Ethernet security via 802.11i with EAP-WPA2, depicted in Figure 4.3, using *hostapd*. In short, 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, visible to clients only as a momentary drop in performance rather than the interface itself going down. We use this to apply blacklisting of clients deemed malicious, such as a client that attempts to communicate directly (at the link-layer) with another, i.e., attempting to avoid their traffic being visible to our home router. We wait until the

[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.

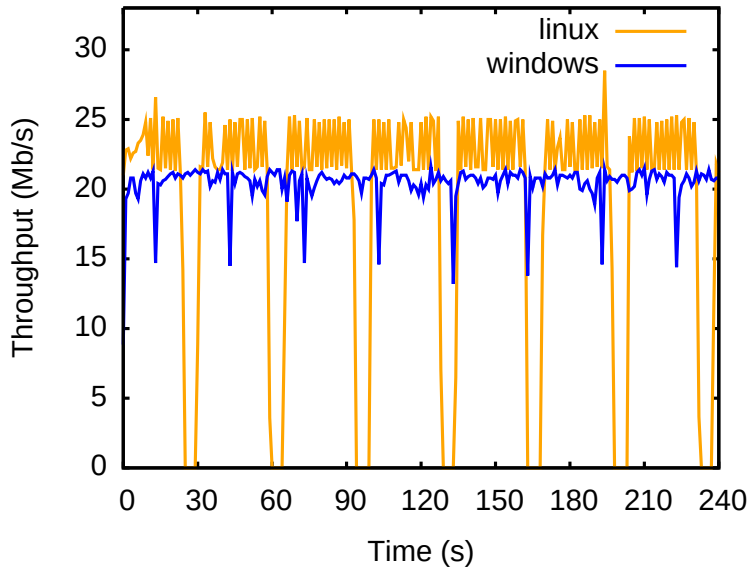


Figure 4.4: Affect on TCP throughput from rekeying every 30s for Linux 2.6.35 using a Broadcom card with the *athk9* module; and Windows 7 using a proprietary Intel driver and card.

rekeying process begins and then decline to install the appropriate rule to allow it 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 impact 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 interim due, to the use of the GTK for

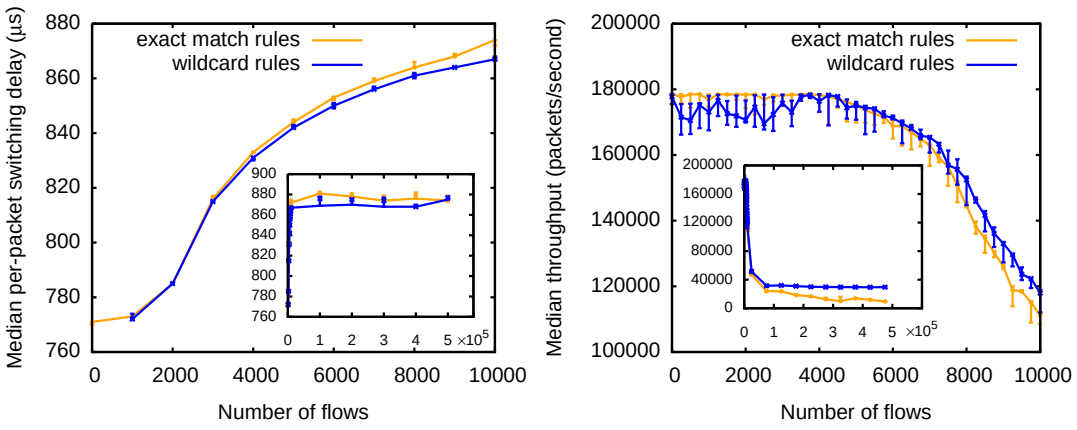


Figure 4.5: Switching performance of Open vSwitch component of our home router showing increasing per-packet latency (LHS) and decreasing packet throughput (RHS) with the number of flows. The inset graph extends the x-axis from 10,000 to 500,000.

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. OpenVSwitch 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 30s 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

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

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. Each test runs for two minutes, generating packets at line rate from a single source to N destinations each in its own 10.2.*.*/30 subnet. As these are stress tests we use large packets (500B) for the latency tests and minimal packets (70B) [1] for the throughput tests, selecting destinations at random on a per-packet basis. Results are presented as the median of 5 independent runs with error bars giving the min and max values.

Figure 4.5 shows median per-packet switching delay and per-flow packet throughput using either exact-match rules or a single wildcard rule per host. Performance is quite acceptable with a maximum switching delay of 560μs and minimum throughput of 40,000 packets/second; 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 4.6 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.

4.4.4 Discussion

Our evaluation shows that OpenvSwitch 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 intentional, accidental or malicious, our home router monitors the number of per-flow rules introduced for each host. If this exceeds a high threshold then the host has its per-flow rules replaced with a single per-host rule, while the router simultaneously invokes user interfaces 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:

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

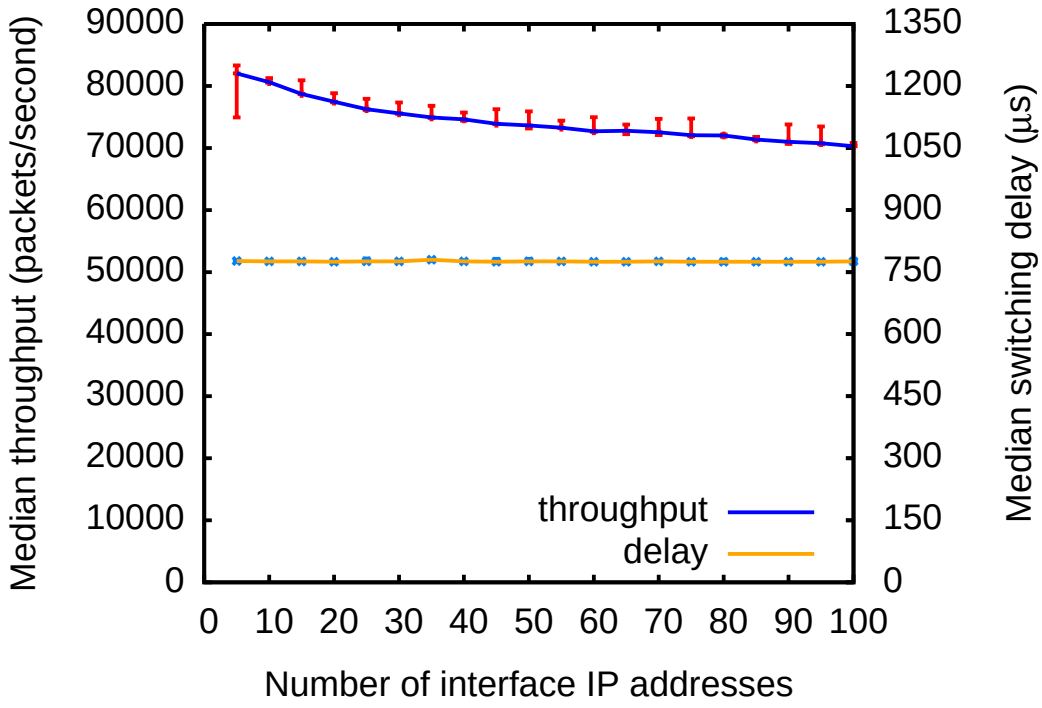


Figure 4.6: Switching performance of Linux network stack under our address allocation policy. Throughput (left axis) shows a small linear decrease while switching delay (right axis) remains approximately constant as the number of addresses allocated to the interface increases.

range of existing client devices; deployed protocols that rely on broadcast/multicast behaviours; and support for IPv6.

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

Table 4.2 shows the observed behaviour of a number of common home-networked devices: in short, all devices operated as expected once permitted access. DNS interception was not explicitly tested since, as an inherently unreliable protocol, all networking stacks must handle the case that a lookup fails anyway. Most devices behaved acceptably when denied access via DHCP or EAPOL, although some user interface improvements could be made if the device were aware of the registration process. The

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.

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 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 this

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

was easy to overcome: simply install a WINS server on the router and advertise it via DHCP to all hosts.

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 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 allocatiocn. 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

[1] OpenFlow aims to provide support in its 1.2 release of the protocol; NOX currently has no support for IPv6; and OpenvSwitch 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.

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 in order that hosts could discover it as the router.

In order to test the functionality of our approach, we setted up a simple hardcoded version of our design. Specifically, we were allocated a public IPv6 /64 prefix to our home router. The machine run the default DHCPv6 server shipped with Ubuntu Natty, configured to allocate addresses from a /120 subnet. Additionally on the router we run the radvd daemon to reply appropriately to ICMPv6 messages. Over this setup we use various IPv6 enabled devices to connect to the internet. From our experiment we verified that Windows and Apple OS had no problem to configure their network through our custom DHCP server configuration. For Linux we identified a buggy implementation of the protocol by the default DHCP client shipped with the distro. This bug was addressed by an alhpa version of the software, which was tested to work as expected.

4.5 Rethinking Home ISP communication

An equally important problem for existing home networks is the limited ability of users to control network resource allocation. This problem can be traced back to two main observations. Firstly, network performance has a multidimensional definition. Performance requirements may include limits over the throughput and latency of a flow and are highly application-specific. The main approach in which the ISP address this problem in a fair way between the users of the network is through the enforcement of monthly or daily traffic caps bt-; vir. Rate limiting queues within the network reduces network utilization, thus reducing packet buffering latencies. Nonetheless, this approach cannot regulate how network flows share available resources, as the end-to-end congestion algorithm impacts significantly the flow resource consumption. Secondly, home network traffic is highly asymmetric. Upload traffic volumes on average are

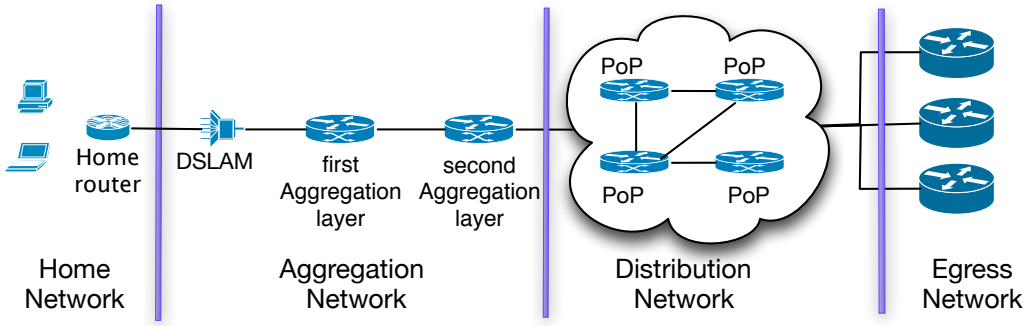


Figure 4.7: The path for each network packet of the home network to the Internet. ISP network can be split in 3 parts: The *Aggregation Network*, the *Distribution Network* and the *Egress Network*.

significantly lower than download volumes. This is due, in part, to the properties of modern home network links, but also due to the nature of modern network applications that function in a client-server manner. In order to control sufficiently resource allocation a system should consider independently both directions of the traffic. This thought is not possible, given the network control limits of the homeowner over only a fraction of the path. Download traffic, for example, is controlled by the home owner only on the last section of the path and thus we cannot develop a sufficient policy enforcing system that has traffic control solely within the home network limits.

In order to better introduce the reader with our architecture, we present in Figure 4.7 a simplified depiction of a broadband ISP network. Although, exact network topologies remain secret, there is a high level design pattern. In the Figure we present the traversed network devices by a packet send from the home network to the Internet. In the Figure, we segment the network topology in four sections, the *home network*, the *aggregation network*, the *distribution network* and the *egress network*. The aggregation network contains the DSLAM that collect the traffic from the user home over the telephone network, the BRAS that ensures that the received traffic is in accordance with the ISP policies and a number of layers of aggregation switches, that multiplex the traffic from the various BRAS on to the distribution network. Traffic from the aggregation is routed internally within the ISP over the distribution network in order to reach either to the egress points of the ISP, or to another aggregation point, if the pattern is destined to an internal service. The distribution network of an ISP consists of a

small number of large routers with high capacity links. Usually at this point network administration use network tunnelling technologies, like MPLS, in order to reduce forwarding information on the routers. The egress network of the ISP consist of large routers that are hosted in AS IXP networks and connect the ISP with peering ASes.

In the described network design a number of research papers have analysed latency and bandwidth performance and 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. The high utilization of this link can be explained by the high ratio of under-provision of such links. The wide adoption of broadband access in households has increased significantly resource requirements and user subscription. Our proposed architecture provides an API that allows users to exercise flow-level control over the available resources of the backhaul link.

In the proposed solution we integrate in the ISP network a mechanism that acquires per-flow information from the users and translates them into appropriate resource allocations. This approach bridges a fundamental gap between the views of the ISP and the users. Users perceive network traffic as an ensemble of flows belonging to a specific application, which applications has a specific prioritisation in the computer interaction process. ISPs, on the other hand, perceive network traffic as an ensemble of network packets aggregated and operated upon the ingress and egress point within the network. Due to the homogenous capping policy on packets from the same home, ISPs tend to collapse any traffic prioritisation of the traffic to or from the home, thus at the moment reducing the ability of the user to control any resource allocations enforced at the router level.

The proposed extension builds around the OpenFlow protocol primitive and requires a switching fabric that can support a rate limiting queue and a prioritisation based scheduling mechanism for each household. We are aware, from unofficial discussion with british ISP network administrators, that current ISP networks have support for similar per-household capabilities in the network. We also need to point out that we don't consider the proposed solution complete and able to provide an holistic system to solve the problem of resource scheduling. We only draw requirement based on observations acquired from relevant user studies and we are trying to map user demands to relevant network functionality. The architecture of the proposed system is described in Section 4.5.1, while, in Section 4.5.2, we present a number of simple

experiment to investigate the behaviour of the proposed system with various types of competing traffic.

4.5.1 User - ISP communication

In the proposed architecture we split the required functionality among three entities in the network: a data collecting daemon on the end-systems of the home network, a policy enforcing daemon on the home network router and an OpenFlow-based mechanism to translate user performance requirements to forwarding policy for the network devices of the ISP.

End-system In order to record the mapping between applications and network flows we have developed a light cross-platform daemon running on the end-systems of the home network. The daemon monitors the connection table of the network stack and inserts information for each new connection in the HWDB database. Each inserted record contains the 5-tuple of the flow and the name of the application that opened the respective socket. Mechanisms to intercept new connection events from the network stack of the kernel are available for most OSes. In our implementation we use *n*etfilter-conntrack net for Linux, *W*indows Filtering Platform win for Windows and *i*pfw for Darwin/MacOSX. These libraries expose similar APIs and allow applications to register callbacks to the kernel network stack, which are invoked every time the state of a record on the connection table changes. In order to match the network tuples with the respective applications, we use the *l*sof command in unix-like systems, while for windows we parse the output of the *n*etstat -p command.

ISP infrastructure In order to handle resource allocation on the bottleneck link we propose the replacement of the switching devices, with OpenFlow-enabled switches that support queue management. The switch control plane is virtualised using the FlowVisor controller Sherwood et al. [2010]. Each home router is running an OpenFlow controller which connects to the FlowVisor instances on the ISP edge network and exercises control only over its own traffic. Specifically, the ISP virtualises the OpenFlow forwarding table and exposes only a subset of the entries to each user. Ad-

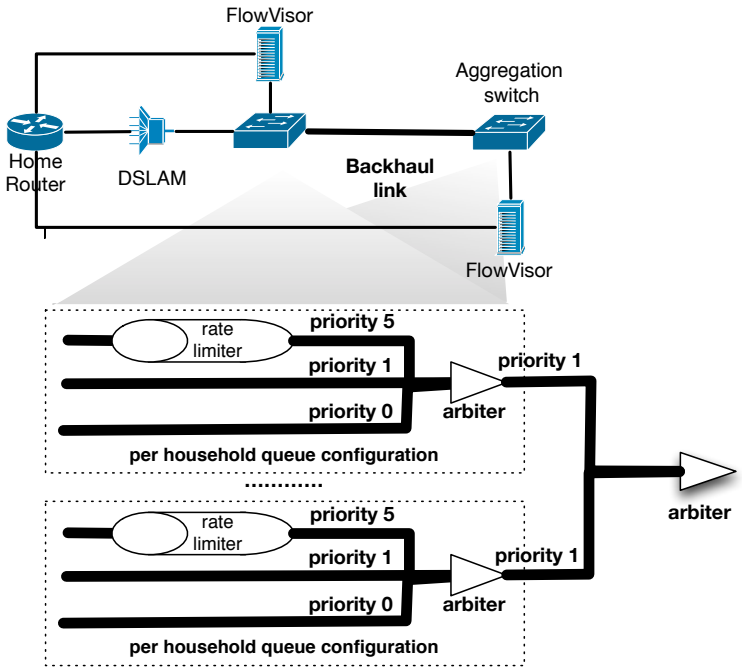


Figure 4.8: Switches handling the backhaul link expose virtual slices to homeowners through a FlowVisor instance. Each switch configures for each household three queue primitives: A *low latency, high priority* queue, a *medium priority* queue and a *default* queue.

ditionally, each network slice exposed to the homeowners informs the controller about traffic with a source or a destination IP address equal to the address of the connection, while flow modification out of this tuple subspace are discarded by the FlowVisor. This abstraction enforces a strong security mechanism; home controllers cannot control or evesdrop traffic out of their scope.

For each switch and for each home in the network we suggest the creation of three priority queues: A *low latency, high priority* (LLHP) queue, a *medium priority* (MP) queue and a default queue. The LLHP queue is configured to have the highest priority from all queues, while it also contains a leaky bucket mechanism configured to rate limit traffic at a rate equal to the minimum guaranteed bandwidth of the connection. The HP queue doesn't enforce any rate limit, but it has a priority which is between the LLHP and the default queue. By default the forwarding table is initialised with a set of flows to forward traffic using the default queue. The home OpenFlow controller can at run-time modify the priority of flows, based on the user configured policy. This capability, though, is constraint by the number of flows that the FlowVisor exposes to each home. The virtualization approach to the ISP network could be further developed in order to provide a new fairer economical model for the home network; Users are not charge solely based on the amount of data, but they can enhance their network performance by purchasing additional flow table entries or higher guaranteed bandwidth on the edge.

Because the system in the proposed design exposes a significant portion of the edge network control to the user, we propose a set of mechanism to nullify the possibility of a home OpenFlow controller to compromise the functionality of the network. Firstly, the reduced number of flows exposed to each home ensures fair utilisation of the forwarding table of the switch. Further, by using the FlowVisor rate limiting functionality per controller, we can fortify the system against control plane DoS attacks. Finally, Flowvisor can be configured to discard flows with an Output action that forwards packet to the incoming port, in order to restrict the ability of a compromised node to create packet loops.

Home Router In our design, we expect the home router to be the rendez-vous point between the policies of the two network. In order to support the state requirements of this functionality we add in the hwdb database three new tables: *Application tu-*

ple, *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 contains timed information on the rate of network applications. 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 on the network consumption of applications. Finally, the Application priority table stores the queue mapping for each application. This table is populated with data from the web interface of the system while the router uses it to map newly arriving flows to queues.

In order to expose in an intuitive way the resource management mechanism, we design a simple web interface, depicted in Figure **??**. Through this interface the user can be informed on the aggregate resource consumption of each application, the time serries of the traffic rate per application and review all priority mappings configured. Additionally, the interface provides user notifications during incidents of underprovisioning of the LLHP queue. Secifically, we use the packet loss parameter from the queue statistics message of the OpenFlow protocol and trigger notification every time packet losses are detected. 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. In detail, for each new arriving flow, after the router has verified that the flow is in accordance with the policy, the controller will lookup the relevant table in the HWDB and will assign the flow in the relevant queue, both on the local instance of the OpenVSwitch, as well as in the remote switches of the ISP. In order to ensure that there is an application mapping for the flow, in case of an incomming connection the switch will establish only the incoming direction if the flow on the local switch and will assign queue only when the outgoing direction of the flow is used. This way we will have ensured that the end-host daemon will have inserted the appropriate information.

4.5.2 Evaluation

We evaluate the proposed architecture using a lab testbed, depicted in Figure **??**. *Home1* and *Home2* emulate the networking activity of an ensemble of Home Con-

nections. 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 Conclusions

This paper has drawn upon previous user studies to reflect on the distinctive nature of home networks and the implications for domestic network infrastructures. Two particular user needs that arose from these studies were for richer visibility into and greater control over the home wireless network, 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.

Simply put, the home is different to standard networking environments, and many of the presumptions made in such networks do not hold. Specifically, home networks are smaller in size, the equipment is physically accessible and access is often shared among inhabitants, and the policies involved are flexible and often dynamically negotiated. Exploiting this understanding allows us to move away from traditional views of network infrastructure, which must be tolerant of scale, physically distributed, and impose their policies on users.

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.

Our exploration suggests that, just as with other edge networks, existing presumptions could usefully be re-examined to see if they still apply in this context. *Do we wish to maintain net neutrality in the home?* Inhabitants do not appear to see network

traffic as equal, often desiring imbalance in performance received by different forms of traffic. *Must the end-to-end argument apply?* Householders understand and exploit the physical nature of their home and use trust boundaries to manage access; we have exploited these resources to explicitly manage the network. *Should communication infrastructures remain separate from the devices that use them?* In the home setting this separation proves problematic as people, ranging from the home tinkerer to the DIY expert, wish to interact directly with the network as they do with other parts of their homes' physical infrastructures. Our exploration suggests use of a range of displays and devices existing not as clients exploiting the infrastructure but as extensions of the infrastructure making it more available and controllable.

Inability to understand and control network infrastructure has made it difficult for people to understand and live with it in their homes. We have developed a home router that both captures information about people's use of the network and provides a point of interaction to control the network. Our initial developments have explored the extent to which residents may be involved in some of the protocols controlling the network; other protocols suitable for modification are under consideration.

Chapter 5

Scalable User-centric cloud networking

In this chapter we explore applications of network control distribution in Cloud and Mobile computing. The work focuses on the problem of information control and distribution between the devices of a user across the Internet; a mechanism we term Personal Cloud. We propose Signpost, an Internet-wide overlay network architecture that establishes Personal Cloud functionality and provides continuous connectivity and controlled security. The architecture consists of daemons running on end-user devices, which can configure and run several off-the-self connection establishing software packages. The proposed architecture employs a distributed negotiation protocol between end-hosts, which scales the control complexity through control distribution.

In order to understand the feasibility and performance of the proposed architecture we develop a strawman implementation, which implements the core control logic of the proposed architecture. Additionally, it integrates support for a number of network connection and notification services. Currently, Signpost supports SSH, OpenVPN, TOR, NAT-punch and Privoxy connection mechanisms, while it can propagate Multicast-DNS notifications across devices.

In this chapter, we present in Section 5.1 the motivation for this work, followed then by the key observations for our design in Section **??**. In Section 5.2, we present the architecture of our strawman implementation and its integration with existing software. Finally, in Section 5.3 we present a number of micro-benchmark tests for our system

and conclude in Section5.4.

5.1 Personal Clouds

In the recent years, the increase in the number of computing devices per user has created a significant information and resource management problem for end-users. In this modern era, the ensemble of the user digital footprint quantums establish a user's abstract digital presence. For example, a users work facet comprises of his work documents and files, while a part of his social experiences can be maps to his collection of digital photos. This digital information is trated by the user as a single entity that he wishes to access through any point of interaction with the digital domain. Unfortunately, the set of personal devices through which a user interacts with his digital presence is large and consists on average of a laptop, a desktop, a tablet, a smartphone and a number of low cost computational units for home entertainment and gaming. These devices tend to offer specific user services and fulfil specific roles, 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, while a number of end-users use smartphones to play audio and video content. In the latter case, the smartphone and the home entertainment system provide similar functionality to the user, require access to the same subset of his digital presence and the user requires a simple abstraction that can replicate his digital footprint between his devices, a functionality which we describe as a *Personal Cloud*.

In order to address these requirements numerous applications and protocols have been introduced that provide functionalities such as remote desktop access, file sharing, remote login, remote printing etc. between devices. Such applications extend existing computer abstractions and integrate mechanisms that allow user to access information or resources between devices. Additionally, the network community has developed a number of mechanisms to reduce the configuration burden of resource sharing mechanisms. Multicast-DNS, Universal Plug and Play and ZeroConf protocols allow a user to seamlessly browse and access available shared services upon connection to a network. Such mechanisms have managed to improve significantly the digital experience in small scale network environments, like the home network, but they face significant difficulties to scale in heavily policed networks or across the Internet. In the rest of this

Section, we discuss the problems arising in Personal Cloud functionality over the Internet (Subsection 5.1.1), present existing approaches to the problem (Subsection 5.1.2) and introduce readers to the Signpost framework (Subsection **??**).

5.1.1 Challenges

Internet is an excellent example of a dynamic system managing to address effectively evolution. In the early days of the system, in order to extend user adoption, its steering committee set simplicity and openness as two fundamental design goals. Due to these design choices, Internet protocols gained rapidly support by a wide range of OSes, while the set of connected entities augmented exponentially. Nonetheless, during that period, the Internet remained a large wide area network, interconnecting research institutes, and the predominant applications provided constraint relaxed asynchronous communication. Through the years though, and as Internet users increased, new applications emerged and highlighted a number of security and performance limitation in the initial design of the system.

In order to address these limitations in a backward compatible manner, a number of network hacks were introduced in the Internet design. One of the most popular hack is the deployment of middleboxes Carpenter and Brim [2002]. Middleboxes violate in a number of ways design principles of the Internet, in order to provide an effective framework to "inject" functionality that addresses design limitation. For example, *NATboxes* handle the IPv4 address space shortage, *WAN optimizers* improve utilization of under-provisioned links, while *firewalls* secure critical networks. Unfortunately, Middlebox deployment redefines to a great extend the Internet abstraction. A number of papers have described their impact: in Honda et al. [2011] authors pinpoint middlebox functionality as a core cause in the ossification of the transport layer, while in Kreibich et al. [2010] authors describe a wide range of protocol functionality that has become suppressed due to middlebox functionality.

describe some efforts to overcome middleboxes. IPv6 can do some of that,but not everything An important impact of middlebox deployment is the reduced connectivity introduced in the Internet. Home networks host a number of hidden devices which remain inaccessible from the Internet due to NATbox functionality, while strict firewall policies in enterprise networks reduce protocol functionality. Personal Cloud

deployment across the Internet is increasingly restricted, as devices are not able to interconnect directly.

5.1.2 Approaches

Personal cloud computing requirements currently experience a mismatch with the functional properties of the Internet. Personal devices usually connect to networks optimised for outgoing connectivity and various components of the Internet are not designed to accommodate well-connected services for every host. NATed networks, that scale connectivity on the edges of the network, require manual configuration in order to host Internet-reachable services, while a number of edge networks, namely mobile and enterprise networks, restrict publicly accessible services on connected hosts for security reasons. In order to overcome these restrictions a number of approach has been proposed by the research community, as well as the industry. We group these solutions in the following two categories:

Decentralised Personal Cloud :

Numerous frameworks have been proposed over the years that enable bidirectional connectivity between devices, using publicly available Internet resource. We are considering in this category tunneling software, like OpenVpn and SSH, and NAT and firewall punching mechanisms, like STUN. Although such mechanisms are effective in a number of scenarios, assumptions and configuration requirements dim them inappropriate for inexperienced users. As we have already discussed in Section 4.1.2, average Internet user is highly improbable to engage in systematic network configuration, if the configuration tasks are long or complex. Establishing connectivity through user-managed mechanisms is not straightforward or guaranteed to succeed. Functionality contains assumptions on the connectivity of the environment and users have to resolve to "try and error" approaches to check which mechanism can be effective in a specific environment, while a number of different network subsystems require prior configuration.

As an example, we describe the steps required by a user to establish connectivity using the SSH service. Before the user is able to connect to a computer using SSH, he has to configure the service on his local machine, the security credentials on the server

and any firewall and NATbox deployed in the network. Additionally, if his public IP is expected to change over the course of a day, he needs to setup a mechanism to access the current IP configuration, like DynDNS. During connection establishment, the user has to run the SSH client, configure the ports he is interested to forward and configure the connecting software to use them. This connectivity is subject to the ability of the user in the remote network to use the SSH protocol on the preconfigure port. In case this is not possible, the user has to resolve to a different service.

cloud-assisted Personal Cloud :

An alternative approach, which has been highly successful in the recent years, uses third party services to establish inter-device connectivity. In this class we consider cloud applications like the Google service suite and Dropbox. These applications provide a simple, intuitive and ubiquitous mechanisms to store and retrieve data in the cloud and define information dissemination policies. Although this approach can be characterised as successful in providing the required functionality, some of the properties are ambivalent and demotivate user engagement.

- *authentication*: Cloud applications provide an effective control framework to disseminate information between devices and users. Users define access policies based on online identities and the service ensures secure information delivery. User authentication mechanisms though introduce privacy concerns, since they can be employed to identify and monitor users. In Krishnamurthy and Wills [2009] authors reports cases of personal information leakage to ad services from Online Social Networks, in order to detect and characterize individuals, and Facebook has openly verified the existence of such services [1].

- *performance*: The availability of large amount of computational resources in current cloud infrastructures provide user acceptable performance, regardless the volume of processed data. This approach though under-utilize the rich computational resources available in end-users device on the edges of the Internet. Two devices connected to the same subnet will experience bloated RTT values when they communicate through a cloud service, while public cloud storage cost and performance is orders of magnitude worse than local network file services.

[1] http://en.wikipedia.org/wiki/Facebook_Beacon

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. *Add dropbox measurement study*

- *cost*: Free cloud services employ a 3-party economical model that subsidizes infrastructural costs through advertisement. Nontheless, cloud services, due to the free nature of the service, provide very weak SLA's. If a part of the service is compromised and sensitive information is leaked, the service provider bears minimum obligation towards affected users. Such costs are not directly observed by the end-user, but they may impact significantly his social and work experience. *report Dropbox problem*

- *Availability*: Cloud services run on well connected infrastructures with a large number of network engineers ensuring security and performance. Any device with Internet connectivity is able to connect to the cloud service without any network configuration. This centralisation of the services over the Internet, introduced a weak link in the service functionality. Devices that can connect directly over a network, will never be able to interconnect and exchange information, if Internet connectivity is unavailable. Two users behind the same firewall will never be able to exchange a file over Dropbox, if the network policy forbids any connection with the servers of the service.

- *G*enerality: Cloud applications develop distributed services optimized for specific functionality. Google Drive provides online document storage, Youtube provides online video hosting and Facebook provides Online Social Network Services. Users are limited on their ability to share information or resources by the offered capabilities of the service provider and there isn't a sole service provider that can support functionality for the complete ensemble of Personal Cloud services.

5.1.3 Reconnecting the Internet

Signpost is a Personal Cloud enabling framework with user controlled security. The system establishes an Internet-wide overlay network between the devices of a user, combining the aforthmentioned approaches. The abstraction relies on a cloud-based

control channel established between devices which can be used to negotiate and establish highly efficient end-to-end connections, using existing decentralized frameworks. In order to ensure functionality under any circumstance, we integrate the control channel with the DNS protocol, a highly available and resilient service.

Signpost has two major design goals. Firstly, the system establishes a user friendly framework that automates end-to-end path configuration between devices. Signpost models the way various existing decentralised mechanisms establish connectivity and encode their configuration and assumption testing in a generic automation framework. Additionally, the system seamlessly integrates support with existing applications. Signpost network paths are exposed in the network layer of the OS, while the API to control the establishment of connection is integrated with the *gethostbyname()* function. Each device is exposed to the users as a domain name and a name lookup for a device triggers the connection establishment mechanism, while traffic send to the returned IP is forwarded through OpenFlow over the established path. Secondly, the system exposes a user-controlled security abstraction, which can provide different security properties on connections between devices. As a result, the user is able to fully control the dissemination of information between his devices, while avoiding privacy sensitive data offload to cloud services.

Add some reference to cloud controller A schematic of the abstraction that our system provides to end-users is depicted in Figure 5.1. In this scenario user Alice, who is at work and uses her smartphone, wishes to access some files on her laptop, which is behind the NATed home router. In order to express her interest to connect to her laptop, Alice needs solely to perform a name lookup for the domain name `laptop.alice.signpost.io`. The name lookup will propagate through the DNS infrastructure to the cloud presence of her Signpost system. The Signpost server will trigger the two devices to try multiple possible connection establishment techniques and setup an end-to-end path between the two devices. Once the server has ensured that a first path is available, it will reply to the initial DNS query with a local IP address which will be routed by the local network stack to the tunnel between the two devices. In parallel, the server will continue recursively to test different tactics, to discover more efficient connection mechanisms.

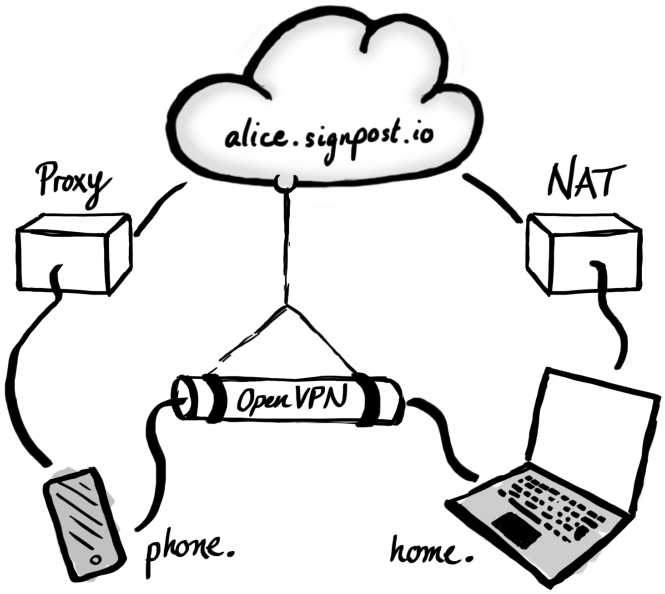


Figure 5.1: A simple example of the Signpost abstraction when the user Alice interconnects a smarthphone with the home computer over the Internet.

5.2 Signpost Architecture

Signpost is an Internet-wide secure inter-device communication system. The system reuses existing Internet protocols and connectivity mechanisms and provides an overlay local network. In Figure 5.2 we present a diagram of the Signpost architecture over different abstractions. In the lower section of Figure 5.2, 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 an OpenFlow switch interface and redirect DNS queries to the embedded DNS resolver. The software consists of three main subsystems: A *Connection Engine* (Subsection 5.2.3) that sets up and manages *Network Tactics* (Subsection 5.2.1 in order to establish end-to-end paths, a local DNS resolver and an OpenFlow-based *Signpost router* (Subsection 5.2.2) that enhances the normal OS routing functionality with Signpost logic. In the middle layer of Figure 5.2, we present the control plane interconnection of Signpost devices. The system relies on an Internet-wide inter-device control channel, which enables connection capability and parameter negotiation. The architecture considers a *Controller Signpost* instance running on a well connected host, which bridges the device control

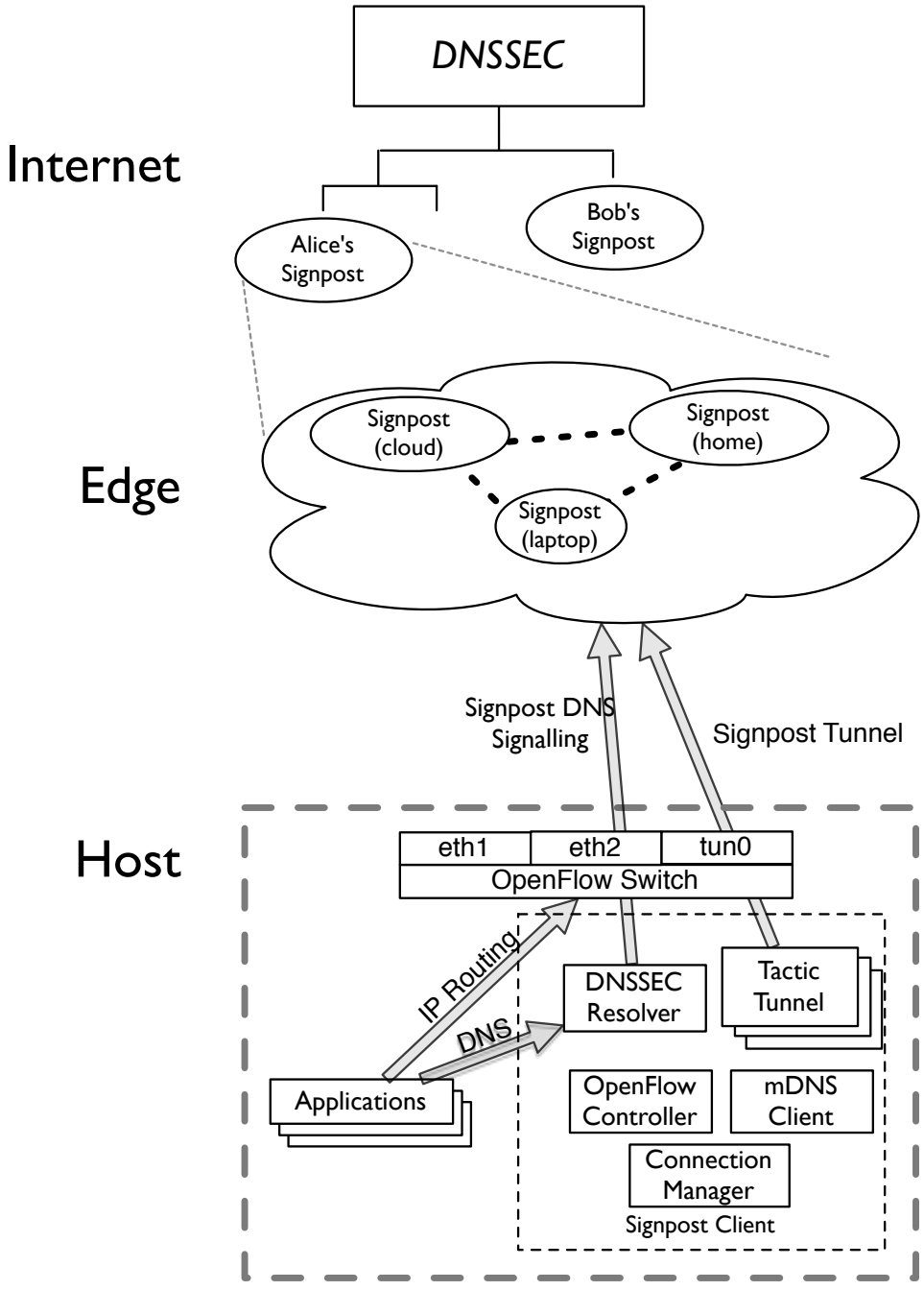


Figure 5.2: Signpost architecture

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.

channel across the Internet. Further, the system can establish connectivity between local devices without the mediation of the Signpost Controller. Signpost clients contain a Bonjour-based discovery mechanism, through which devices can establish ad-hoc secure and authenticated paths. 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-enable domains; a name resolution expresses the interest of a user to establish an end-to-end path (Subsection 5.2.4). For the rest of the section we present in depth the details of the Signpost system.

5.2.1 Network Tactic

Currently the network community offers a wide selection of software to establish connectivity between network devices. In Table 5.1, we present a small survey of such

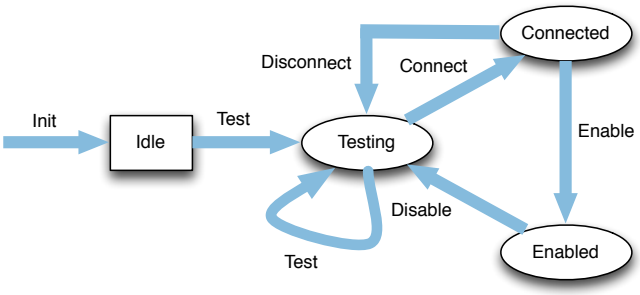


Figure 5.3: Signpost tactic lifecycle

mechanisms along with their network requirements and security properties. From the data of the table, we note the high diversity between connection properties. For example, a significant subset of the mechanisms abuse application layer protocol functionality in order to establish IP connectivity, thus allowing users to bypass strict network policies. An equally significant subset of the mechanisms focus on encryption and privacy enhancement of end-to-end Internet paths, while a third class of these mechanisms enables connectivity through simple service advertisement. Further, the mechanisms vary significantly on the network layer they operate and the type of connection they provide. Tunnels provide connectivity on a specific port of the transport layer, or introduce an overlay network on the network layer. The majority of the tunelling mechanism functions over UDP or TCP protocol, while there are protocols that function over lower layer protocols, like ICMP. Finally, authentication of users is not uniform. Approaches vary from user-based authentication, using either passwords or certificates, to simple passphrase checks, while a subset of the protocols doesn't support any authentication.

Due to the wide and diverse range of available mechanisms, we model their functionality in terms of Signpost through a simple abstraction, which we term *Network Tactic*. A sufficiently generic specification of this abstraction is core for the establishment of an automated connectivity platform. This abstraction enables modularization during the integration of new mechanism, while enabling the development of algorithms that optimize specific indexes on the exploration of the optimal combination of tactics to fulfil a user connection policy.

Each tactic in Signpost 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, while executing the testing logic of the tactic. The test method performs tactic specific testing in order to detect the limitations in network connectivity and the configuration required in order to establish connectivity. If testing is successful, the tactic can progress to the connect state which will configure an end-to-end path. Once the end-to-end path is setted up, then the Tactic can progress to the enabled state and forward packets to the end-to-end path. Finally, the tactic automata provides methods to backtrack from each state and clear stored state. In order to avoid packet reordering, Signpost permits parallel existence of multiple connected tactics for a set of devices, but a single tactic can be enabled at any point. *Mention that tactic is able to control OpenFlow also*

In term of modules implementation, Signpost achieves control distribution through the definition of a generic model to split functionality between the Signpost controller and client for a specific tactic. The functionality of a Signpost tactic is split logically in two layers: the Southbound layers, which implements low level tactic operations, and the Northbound module, which translates the tactic abstraction into low level operations. The Northbound module logic is executed by the Signpost controller, while the Southmodule module logic is executed by the Signpost clients. The two layers of the tactic, communicate over the control channel using a tactic specific protocol.

As an example of this functionality split, we present the Test methof of the OpenVpn tactic, an IP tunneling system that uses certificate-based authentication and functions over TCP or UDP. In the test section of the Southbound layer, the tactic exposes two main functionalities: init an OpenVpn server with default configuration and test if an OpenVpn server is accessible on a given IP and port. The Northbound layer of the tactic will instruct all devices participating to initiate a server instance and test connectivity to the other end of the link. The first client that will return with a successful result will become a client, while if the test request times out for both devices, then the server will initialise an OpenVpn server and instruct the devices to test connectivity through the cloud.

Discuss weight of tactics

5.2.2 Forwarding

In order to add seamless Signpost support in existing applications we choose to enable Signpost integration at the network layer. As a result, a Signpost cloud is abstracted as a local subnet and persistent local IPs are allocated to devices. In order to modify the forwarding logic of the end-host, we add a requirement for an OpenFlow switch running on the local system which will be connected to the embedded OpenFlow controller of the Signpost software.

The forwarding logic of Signpost avoids any interference with the normal network functionality of the system. Signpost is responsible to handle flows that are under the Signpost local subnet and the system at start up configures appropriately the routing table of the host, as well as the OpenFlow flow table of the switch with static entries. For flows that interconnect Signpost devices under the Signpost subnet, the system delegated their control to the respective enabled tactic which can exercise control either actively or pro-actively through a simple event driven model. Additionally, the controller enables tactics to inject packets in the network, through the OpenFlow protocol, while they are also able to install proactively OpenFlow flows that can modify normal network routing functionality. Tactic developers have to be careful with this control delegation and install flows that affect solely tactic traffic. Finally, in order to reduce broadcast traffic notification in the control channel, we implement a simple ARP cache as part of the OpenFlow controller. The ARP caches replies with the MAC address fe:ff:ff:ff:ff:ff on every ARP request for IP addresses within the Signpost subnet. Tactics are responsible to exercise network level forwarding control.

5.2.3 Connection Manager

Signpost is able to establish multiple end-to-end paths between devices through tactic synthesis. Unfortunately, multipath connectivity is not supported by popular transport protocols and newer protocols like SCTP and multipath TCP, that support multipath connectivity, are not available in production applications. Multipath functionality in Signpost could be integrated in existing transport protocol through careful OpenFlow flow manipulation, but because performance is not homogenous between paths, packet reordering may occur and reduce network stack functionality. As a result, Signpost we establish and use a single end-to-end path between any two devices. Path search and

establishment is encapsulated in the Connection Manager Subsystem.

Signpost path selection mechanism considers two parameters: tunnel performance and security policy. In terms of tunnel performance, we use a weight mechanism and represent performance as a positive integer value, defined manually by the developer. We use a set of simple rule of thumbs to define tactic performance weight. Tactics that include the Controller in the forwarding path, are considered less efficient than those establishing direct connectivity, while tactics that run over connectionless protocols, like UDP, are preferred over protocol that run over connection-oriented protocols, like TCP. In cases of tactic synthesis, the performance index is computed as the sum of the performance of the partial tactic. This simplistic approach appears to be sufficient at the moment, but more complex tactic specific mechanisms can be employed, using active measurement performance estimates. In terms of security policy Signpost exposes a simple interface. The policy is expressed through a configuration file and users specify security requirements on a per domain basis. Signpost architecture considers three security properties: *encryption*, *authentication* and *anonymity*. The encryption property establishes an end-to-end path with a strong cryptographic cipher applied on both ends, the authentication property applies on mechanisms that employ strong authentication during the establishment of an end-to-end path, while the anonymity property applies on tactics that obfuscate user identity details from the Internet service provider.

Signpost path search user a simple width-first search algorithm over the network tactic abstractions on the controller of the Signpost Cloud. The search is initiated by the connect method of the Manager module, with parameters the name of the devices and a list of connection properties. The Manager is then responsible to establish the end-to-end path and return once a first path is available.

The logic of the search algorithm is pretty simple. During a connection request the system spawns a thread for each available tactic. The tactic thread tests and, if successful, connects the two end-points. If the connection was successful and either there is no other Tactic enabled or the currently enabled tactic has a higher performance score, then the Manager will progress the tactic state machine to the Enabled state. Otherwise, the tactic is reset back to the testing state. In order to support synthesis of multiple tactics, if the connected tactic doesn't fulfil the security policy, the Manager will recursively enable tactics on top of the established path. Once the first tactic during a search becomes enabled, the Manager will return a positive value to the caller,

while the Manager will continue recursively to search for better tactics. The recursion terminates if the remaining tactics have a higher performance weight that the currently enabled tactic, or if the depth of the search is higher than three. Using this algorithm, the Manager can provide a quick reply to a path request, while in the background it searches for optimal tactic combinations.

5.2.4 Effectful Naming

The majority of Internet-connected devices are essentially anonymous from a network perspective, and assigned transient names (e.g. via DHCP). A fundamental requirement for the Signpost architecture is to assign stable names to each device, and provide a secure mechanism to resolve these names into concrete network addresses. Signpost naming functionality is established through the DNS protocol Mockapetris [1987] for two main reasons. On one hand, DNS is an effective solution for the problem we try to address. DNS is a widely deployed and accessible service across the Internet, which is never blocked by the local network and provides a sufficient mechanism for bi-directional authentication. In addtion, the DNS service is an excelent mechanism to intercept user connectivity intentions. Internet naming service is a delay-tolerant mechanism for applications to express interest to connect to a service. DNS domain names provide a naming representation of Internet services decoupled from the network layer technology, while the naming format is a good match to spoken language.

Domain naming follows a simple organisation model which though provides very good scaling properties. Every domain name consist of a sequence of name token which are 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) which define information for the specific domain name like network addresses, naming aliases, service information and many more. In order to scale the naming service efficiently across the Internet, the protocol provides a specific RR type, the NS record, which delegates control for a domain to a specific set of DNS servers. Querying the domain tree for a specific RR requires at least the address of a DNS server. The DNS client can then follow service redirections in order to find a server responsible for the requested domain. Additional, the explicit caching mechanism of the service reduces significantly the load on naming servers.

Secure Names For All Internet Users In the initial definition of the 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, which ascribe as DNSSEC. DNSSEC defines several extra RR types Arends et al. [2005] used to provide a signing chain that can authenticate DNS records. In essence, a zone signs its authoritative RRsets using its private key, and then publishes its public key via a DNSKEY RR to allow resolvers to validate the RRset signatures. Following the signing of the root zone, and certain top level domains beneath it, a chain of trust is formed back to the root, whereby a given resolver can be certain that the response it has received to a query *is* the correct response from the authoritative DNS server for the name in question. 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.

In terms of the Signpost system, we have been delegated control of the `.signpost.io` domain and registered a signing key with the `.io` domain. For each user of the Signpost system, we delegate and authenticate control to a subdomain and redirect DNS queries to the Signpost Controller of the user personal cloud, where the user can register its devices. As an example with reference to Figure 5.2, user Alice is granted control of the domain `alice.signpost.io` where she can register her laptop under the domain name `laptop.alice.signpost.io`. By running a Signpost server, an individual has a globally accessible authenticated public identity on the Internet via their 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, defined in Arends et al. [2005], provide a mechanism to authenticate the channel from the server to the client. In terms of Signpost, we are also interested to enable an authenticated channel from the client to the server. This will permit to 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], which functions as a signature on a request. The record was introduced in

[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]*

order to allow authorized clients to update RR records on an authoritative server, while it permits a client also to point to the signing entity, in order to fit authentication with the DNSSEC key structure.

Fitting DNSSEC in Signpost In terms of Signpost design, we modify DNS functionality both on the client and the controller of the architecture. On the controller we develop a programmable DNS server that functions as an authoritative server for the user domain. The server provides DNSSEC access to any host in the Internet for the SOA, DNSKEY and NS records of the domain, signed with RRSIG records on the fly. Additionally, the server can verify SIG(0) records from queries and translate signed requests from Signpost clients in Connection Manager requests. A request for records of A for domain name `laptop.alice.signpost.io`, signed with a SIG(0) record with a key from host `desktop.alice.signpost.io`, will be translated in a connection request between Alice's laptop and desktop device to the Connection Manager. In order to enforce liveness of RR record in the Internet, we set a zero TTL value on all RR records, thus disabling any DNS caching.

In the client side of the Signpost architecture, we have developed a local Signpost-aware DNS resolver. The resolver enhances the typical resolver functionality by signing Signpost connectivity request. For normal DNS queries, the resolver will fetch recursively requested records. For RR requests under the Signpost domain, the resolver augments the query with a SIG(0) record signed with the private key of the device.

discnnected Signpost functionality

Using the DNS protocol we are also able to ensure the functionality of the system when devices are disconnected from the Signpost Controller, while connected in the same network. Signpost uses DNS-based service discovery (DNS-SD) Cheshire and Krochmal [2011b] in order to enable device local connectivity. DNS-SD is a specification of DNS RR organisation which enables efficient service advertisement and browsing and along with Multicast-DNS Cheshire and Krochmal [2011a] they establish an efficient local service discovery mechanism. The combination of DNS-SD and multicast-DNS is currently a widely used mechanism for service discovery in local networks and supported by most available operating systems. Signpost registers and advertises a service record for the Signpost service with name _sp._tcp.local in the local network with a target the domain name of the device, along with an RRSIG RR and

the DNSKEY RR of the device, signed with the private key of the user. Using these records a Signpost client is able to verify a destination Signpost service and establish a control channel. The device will use the service advertisement information when a name lookup is performed. During a name lookup, the destination device will function as a Signpost Controller responsible for the specific device only and employ all the logic described previously.

5.2.5 Security and Key Management

In order to bootstrap device authentication, Signpost uses a public key cryptography mechanism integrated with the DNSSEC protocol. Signpost employs a key hierarchy, which enables the system to control and revoke trust on device keys in real time. Our key hierarchy relies and extends the key hierarchy defined by the DNSSEC protocol. For a Signpost cloud the user should construct at least two keys, A *Zone Signing Key (ZSK)* and a *Device Signing Key(DSK)*. ZSK is used to sign DSK and a signed hash of the ZSK is served by the signpost.io domain servers, in order to add the key in the global DNSSEC keychain. DSK is used by the controller to sign Device Keys and the key is signed by the ZSK. This differentiation of user keys permits a Personal cloud to have a persistent anchor in the DNSSEC key chain, while the DSK allows the cloud to frequently update his key trust, in fixed weekly basis or when a key compromise is detected, without having to rely in the DNSSEC infrastructure to propagate the updates to other servers. Each device registered with the cloud must construct a private Device Key when it first joins the system and add the public key to the Controler device key cache over a trusted channel. The public key will be signed by the DSK of the Cloud and shared as a DNSKEY RR by the controller. A device of a Signpost cloud requires only an anchor in the global DNSSEC key infrastructure and it can easily verify the validate of any Signpost key.

5.3 Evaluation

In this section we analyse the performance of the Signpost system and its impact in the functionality of traditional Personal cloud applications. We, present the implementation details of the system (Subsection **??**), measure the performance of the Signpost

tactics over the Internet (Subsection **??**) and present a small scale study of the functionality of current application over the Signpost system.

5.3.1 Signpost implementation

Signpost is implemented predominantly in Ocaml. The code reuses a number of available protocol libraries written in Ocaml. Signpost uses ocaml-dns for DNS server and client implementation, cyptokit library for cryptographic key manipulation and ocaml-openflow library for OpenFlow controller functionality. Signpost also uses a portion of C code code to implement binding with the OS routing stack. Signpost provides support for a wide range of platform. We have managed to run Signpost successfully under Linux and Android, using the OpenVSwitch switch implementation, and under MacOSX, using the userspce switch implementation provided by the ocaml-openflow library.

Signpost provides support for the following tactics:

- *Direct*: The tactic enables communication path between devices over the network without any tunneling mechanism. The functionality of the tactic provides a mechanism to test direct connectivity between the two devices over a number of well-known ports and if successful it will insert a single OpenFlow rule to translate Signpost IP addresses to the respective network addresses.

- *OpenVpn*: The tactic enables communication paths using the OpenVpn tunneling mechanism. During connection, Signpost devices will 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 will generate transient private keys which will be signed by the user private key, while the tactic will inject in the trusted certificate keychain a key certificate of the destination device signed by the private key of the device. The tactic configure the OpenVpn software to expose connectivity over an ethernet TUN/TAP device, which is added under the OpenFlow switch control and normal OpenFlow packet forwarding rule establish full bidirection connectivity. *Mention OpenVPN ARP cache*

- *SSH*: The tactic enables connectivity of the system using the SSH protocol. For

this tactic we configure and run a separate ssh daemon on every Signpost device. The daemon runs on port 10000 and permits key-based authenticated connectivity for tunneling. User authentication is ensured by proper manipulation of authorized keys on the server configuration. Signpost clients append device public keys in the authorized_key file in an ad-hoc manner, while clients are instructed to use the private key of the device upon connection. SSH program is configured to expose TUN/TAP ethernet interfaces between devices, which are added under the control of the OpenFlow switch.

- *Privoxy*: This tactic enables HTTP request anonymization using the privoxy HTTP proxy. In order ot achieve this, we have predefined a strict configuration of the privoxy software which strips all HTTP headers that may leak personal information of the user. In order to establish connectivity, the tactic handles TCP flows in a proactive manner. For each SYN packet, the tactic injects two flows that forward data from the application to the loopback device and to the listening port of the Privoxy proxy and reverse.

- *Tor*: The tactic enables connectivity between devices over the Anonymized network of Tor. Tor client software exposes a SOCKS proxy on the local machine which can provide TCP connectivity to flows over the Tor overlay network. In order to enable bidirectional connectivity over the Tor network, the tactic uses the hidden service functionality of the system. A node in Tor is able to expose listening ports. In order to achieve this, the system uses random domain names under the .onion domain to address nodes and it takes advantage of the socks protocol capability to embed name lookups in SOCK connection requests. Upon the request for connection over the Tor network, the Signpost client will negotiate with the remote device its domain name under the Tor abstraction. Upon a SYN packet transmission of the application to the remote device, 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 and advertise a zero window in order to suppress any further data transmission from the application. Once a SOCKS response is returned by the SOCKS 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 client will inject a TCP RST if the SOCKS response was unsuccessful.

- *DNS-SD*: This tactic is used to advertise services provided by the Signpost devices. DNS-SD is a popular mechanism by current OSes to advertise available host services in the local network. The tactic doesn't provide any path establishing functionality to applications, but it focuses to provide a mechanism to aid the advertisement of host functionality. The functionality of the tactic will intercept DNS-SD service advertisement and propagate them over the control channel, to the other devices of the Cloud. Each Signpost client is responsibled then to inject DNS-SD multicast packets over the local loopback device of the OS and propagates information in the local ZeroConf daemon.

- *NAT-punch*: This tactic uses simple packet injection mechanisms in order to allow devices to bypass NAT boxes. The functionality of this tactic is limited and covers only the cases of Full-cone NAT, Address-restricted NAT and Port-restricted NAT, and requires that the NAT functionality doesn't perform any stateful packet filtering. Nat punching functionality is achieve by using the Controller as an intermediate service that can infer the port mappings configuration of the NAT-box. On a TCP connection, the server will intercept the SYN request, propagate the important TCP header parameters over the control channel to the destination device and in parallel will send a SYN packet from the device to the Controller in order to infer the port mapping applied by the device. On the destination device the Signpost client will generate a SYN packet with similar values to the inital SYN packet and send the packet to the local listening service over the loopback device as well as the Controller of the Cloud. During this processing, both end of the TCP connection hold the connection idle with a zero window. Once the exact port-mapping is inferred on the controller, the information is propagate to both devices, which will insert appropriate OpenFlow commands to manipulate source and destination port renumbering and notify the end-points to resume data transmission using a ACK with a non-zero window size.

5.3.2 Tunnel Evaluation

5.3.3 Application compatibility

5.4 Conclusions

- This is an elementary approach for Personal Cloud computing.
- A number of issues remain unandressed but can easily fit in the Signpost architecture
- extending the policy mechanism, we can integrate in a Personal cloud devices from other users. Need though to develop a more refined policy that will enable the user to control access from devce outside of its personal to a subset of the available services.
- Tighter DNS integration of the control protocol.
-

Chapter 6

My Conclusions ...

Here I put my conclusions ...

References

Baremetalos. URL http://www.returninfinity.com/baremetal.html. 30

Broadband usage policy. URL http://bt.custhelp.com/app/answers/detail/a_id/10495/~/broadband-usage-policy. 62

Cbench: controller benchmark. URL http://docs.projectfloodlight.org/display/floodlightcontroller/Cbench. 27

Emulab: Network emulation testbed. URL http://www.emulab.net. 27

cooperative threads library for ocaml. URL http://ocsigen.org/lwt/. 29

libnetfilter conntrack project. URL http://www.netfilter.org/projects/libnetfilter_conntrack/index.html. 65

OFLOPS. http://www.openflow.org/wk/index.php/Oflops. 15

Planetlab: An open platform for developing, deploying and accessing planetary-scale services. URL http://www.planet-lab.org. 27

Virgin media cable traffic management policy. URL 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. 62

Windows filtering platform. URL http://msdn.microsoft.com/en-us/library/windows/desktop/aa366510.aspx. 65

Openflow switch specification (version 1.0.0). www.openflow.org/documents/openflow-spec-v1.0.0.pdf, December 2009. 16, 18, 22

The snac openflow controller, 2010. http://www.openflow.org/wp/snac/. 14

B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, H. Levkowetz, and Ed. Extensible Authentication Protocol (EAP). RFC 3748, IETF, June 2004. 57

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. 64

A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15(2), June 2005. 50

Roy Arends, Rob Austein, Matt Larson, Dan Massey, and Scott Rose. Rfc 4034: Resource records for the dns security extensions. *Internet Engineering Task Force*, 2005. 86

Patrik Arlos and Markus Fiedler. A method to estimate the timestamp accuracy of measurement hardware and software tools. In *PAM*, 2007. 13

Giacomo Balestra, Salvatore Luciano, Maurizio Pizzonia, and Stefano Vissicchio. Leveraging router programmability for traffic matrix computation. In *Proc. of PRESTO workshop*, 2010. 24

Steven Bauer, Robert Beverly, and Arthur Berger. Measuring the state of ecn readiness in servers, clients,and routers. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, IMC '11, pages 171–180, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-1013-0. doi: 10.1145/2068816.2068833. URL http://doi.acm.org/10.1145/2068816.2068833. 8

A. K. Bhushan. RFC 354: File transfer protocol, July 1972. 3

A. K. Bhushan, K. T. Pogran, R. S. Tomlinson, and J. E. White. RFC 561: Standardizing network mail headers, September 1973. 3

A. Bianco, R. Birke, L. Giraudo, and M. Palacin. Openflow switching: Data plane performance. In *IEEE ICC*, may 2010. 16

S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. RFC 2475: An architecture for differentiated services, December 1998. 7

J BORDER, M KOJO, J GRINER, et al. Rfc3135: Performance enhancing proxies intended to mitigate link-related degradations. June 2001. 7

Stewart Brand. *How Buildings Learn: What Happens After They're Built*. Penguin, 1995. 44

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. 46

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. 35

Brian Carpenter and Scott Brim. Rfc 3234: Middleboxes: Taxonomy and issues. 2002. 73, 86

Stuart Cheshire and Marc Krochmal. Rfc 6762: Multicast dns. *Work in Progress*, 2011a. 87

Stuart Cheshire and Marc Krochmal. Rfc 6763: Dns-based service discovery. 2011b. 87

M. Chetty, J.-Y. Sung, and R.E. Grinter. How smart homes learn: The evolution of the networked home and household. In *Proc. UbiComp*, 2007. 44

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 *Proceedings 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. 44, 46, 68

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. 43

Lucas Di Cioccio, Rennate Teixeira, and Catherine Rosenberg. Characterizing home networks with homenet profiler. Technical report, Technicolor, 2011. 41

I Cisco. Cisco Visual Networking Index: Forecast and Methodology, 2011–2016. *CISCO White paper*, 2012. 4

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. 3

D. Cohen. RFC 741: Specifications for the network voice protocol (NVP), November 1977. 3

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. 15

A. Crabtree, T. Rodden, T. Hemmings, and S. Benford. Finding a place for ubicomp in the home. In *Proc. UbiComp*, 2003. 44, 47

J D Day and H Zimmermann. The OSI reference model. *Proceedings of the IEEE*, 71 (12):1334–1340, 1983. 4

S. Deering and R. Hinden. RFC 2460: Internet Protocol, Version 6 (IPv6) specification, December 1998. 7

Ruby Roy Dholakia. Gender and it in the household: Evolving patterns of internet use in the united states. *The Information Society*, 22(4):231–240, 2006. 3

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.

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. 5, 64

R. Droms. Dynamic Host Configuration Protocol. RFC 2131, IETF, March 1997. 52

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. 1

Donald E Eastlake. Rfc 2931: Dns request and transaction signatures (sig (0) s). 2000. 86

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. 43

FCC. A Report on Consumer Wireline Broadband Performance in the U.S. URL http://www.fcc.gov/measuring-broadband-america/2012/july.

R.E. Grinter and W.K. Edwards. The work to make the home network work. In *Proc. ECSCW*, 2005. 43

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. 14

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. 35

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 2009. 13, 26

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. 42

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. 8, 73

Junxian Huang, Qiang Xu, Birjodh Tiwana, Z. Morley Mao, Ming Zhang, and Paramvir Bahl. Anatomizing application performance differences on smartphones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, MobiSys '10, pages 165–178, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-985-5. doi: 10.1145/1814433.1814452. URL http://doi.acm.org/10.1145/1814433.1814452. 5

ITU. The world in 2011: Ict facts and figures, 2011. http://www.itu.int/ITU-D/ict/facts/2011/material/ICTFactsFigures2011.pdf. 1

Lavanya Jose, Minlan Yu, and Jennifer Rexford. Online measurement of large traffic aggregates on commodity switches. In *Proc. of the USENIX HotICE workshop*, 2011. 24

Karthik Kanan, Jackie Rees, and Eugene Spafford. Unsecured economies: Protecting vital information. *Red Consultancy for McAfee, Inc*, 2009. 6

S. Kent and R. Atkinson. RFC 2401: Security architecture for the Internet Protocol, November 1998. 6

Mathias Klang and Andrew Murray. *Human Rights In The Digital Age*. GlassHouse, 2005. 3

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 . URL http://yuba.stanford.edu/openflow/documents/openflow_deployment_journal_paper_aug2012.pdf. 20

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. 42, 73

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. 75

A Kuzmanovic, A Mondal, S Floyd, and K Ramakrishnan. Rfc 5562: adding explicit congestion notification (ecn) capability to tcps syn, June 2009. 7

J. C. R. Licklider. Memorandum For Members and Affiliates of the Intergalactic Computer Network, April 1963. URL http://www.kurzweilai.net/memorandum-for-members-and-affiliates-of-the-intergalactic-computer 3

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. 34

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. 43

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. vi, 45

D L Mills and H W Braun. The NSFNET backbone network. *ACM SIGCOMM Computer Communication Review*, 17(5):191–196, 1987. 3

CVNI Mobile. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2011–2016 . *White Paper*, 2012. 4

P. V. Mockapetris. RFC 1034: Domain names — concepts and facilities, November 1987. 85

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. 48

Jad Naous, David Erickson, G. Adam Covington, Guido Appenzeller, and Nick McKeown. Implementing an openflow switch on the netfpga platform. In *ANCS*, 2008. 17

R. Olsson. *pktgen*, the linux packet generator. In *Proc. Linux Symposium*, 2005a. 58

R. Olsson. pktgen the linux packet generator. In *Proceedings of Linux symposium*, 2005b. 15

Joshua Pelkey and George Riley. *Distributed Simulation with MPI in ns-3*. March 2011. 34

Justin Pettit, Jesse Gross, Ben Pfaff, Martin Casado, and Simon Crosby. Virtualizing the network forwarding plane. In *DC-CAVES*, 2010. 16, 17

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. 42

E Rescorla. Rfc 2631: Diffie-hellman key agreement method. *Internet Engineering Task Force*, 1999. 86

T. Rodden, A. Crabtree, T. Hemmings, B. Koleva, J. Hunble, K.-P. Akesson, and P. Hansson. Between the dazzle of a new building and its eventual corpse: assembling the ubiquitous home. In *Proc. ACM DIS*, 2004. 44

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. 44

J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2(4):277–288, November 1984. ISSN 0734-2071. doi: 10.1145/357401.357402. URL http://doi.acm.org/10.1145/357401.357402. 7

Aman Shaikh and Albert Greenberg. Experience in black-box ospf measurement. In *ACM IMC*, 2001. 22

E. Shehan and W.K. Edwards. Home networking and HCI: What hath God wrought? In *Proc. ACM CHI*, 2007. 45

E. Shehan-Poole, M. Chetty, W.K. Edwards, and R.E. Grinter. Designing interactive home network maintenance tools. In *Proc. ACM DIS*, 2008. 43, 45

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*, 2010. 13, 65

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. 43

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. 48, 50

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. 7

Ajay Tirumala, Feng Qin, Jon Dugan, Jim Ferguson, and Kevin Gibbs. Iperf: The tcp/udp bandwidth measurement tool. 2005. 69

P. Tolmie, A. Crabtree, T. Rodden, C. Greenhalgh, and S. Benford. Making the home network at home: digital housekeeping. In *Proceedings ECSCW*, 2007. 44, 45, 47

Amin Tootoonchian, Monia Ghobadi, and Yashar Ganjali. OpenTM: traffic matrix estimator for openflow networks. In *PAM*, 2010. 24

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. 28

Paul Weissmann and Srini Seetharaman. How mature is OpenFlow to be introduced in production networks. URL http://changeofelia.info.ucl.ac.be/pmwiki/uploads/SummerSchool/Program/session_004.pdf. 13

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. 76

Kok-Kiong Yap, Masayoshi Kobayashi, David Underhill, Srinivasan Seetharaman, Peyman Kazemian, and Nick McKeown. The stanford openroads deployment. In *Proceedings of ACM WINTECH*, 2009. 26

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. 42

Minlan Yu, Jennifer Rexford, Michael J. Freedman, and Jia Wang. Scalable flow-based networking with difane. In *ACM SIGCOMM*, August 2010. 13