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

This thesis states that the best way currently to evolve computer networks is through the evolution of the control domain of networks. We state that in order to make control more effective in a computer network, we need to evolve control on the edges, and expose an API to users and applications in order to allow them to express their interest more explicitly.

The evolution of human communication needs has been radical in the recent years and Internet has evolved as the main medium. It has become vital from many aspects of society, while Internet accesibility is slowly recognised as a fundamental human right. Network engineering hasn't been fully capable to support this development through sufficient innovatio. Network perfromance requirements are enhanced and modern networks have become highly complex, as well as network performance requirements. Although, link rates have inceased significantly, the complexity of modern networks hardens the optimization task.

In order to

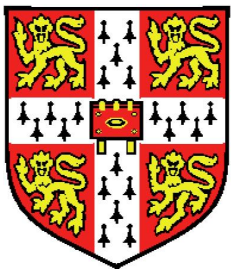
Keeping in accordance with the end-to-end principle of computer networks, a approach would be to develop more efficient protocols. Unfortunately, the requirement for fast connectivity at low cost, has assimilate to the network a number of strong assumtpions, that make it impossible to develop and propose new network protocol that address aforthmentioned problem. An alternative approach to the problem is to provde evolution through the control plane. Such approaches have been explored in the past without a lot of adaption. A recent development in the field is called *SDN* and gains a lot of interest from the comunity.

In my thesis, I will firstly present a set of evaluation platform and results that try to understand the impact of the SDN paradigm, and especially

its popular implementation *OpenFlow*. The result show that the protocol implementation are not yet sufficiently mature to be deployed across the network. Although, software implementations of the protocol are exceptionally efficiently.

This observation drives my exploration on the possibilities of deploying OpenFlow in the edge of the network, close to end-users.

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

This thesis states that the best way currently to evolve computer networks is through the evolution of the control domain of networks. We state that in order to make control
2, more effective in a computer network, we need to evolve control on the edges, and expose an API to users and applications in order to allow them to express their interest more explicitly.

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

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

27, add some pointers.

43, user engagement with network and required functionality

43, expressing user collaboration on the social aspect. how user share their network?

44, Mismatch of user perception to real properties of the network. e.g. what are the limits of the local network?

Chapter 1

Introduction

Internet has become the predominant mode of communication in the modern societies 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. Regardless the vital role of computer networking in our life, its strong backwards compatibility ties create a gap on our ability to evolve functionality in order to fulfil current resource short-term requirements. As a result, although the social setting requires novel functional properties from its global network, it is rather difficult to provide it, without disconnect a portion of it.

My work focuses on the evolvability problem of modern networks. The key idea of this work focuses on ways to evolve computer network functionality through the control plane. 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 performance functionalities to the inter-connecting fabric. Such control mechanism should address the requirements of the deployment environment while taking advantage of its distinct properties. Edge networks provide an excelent deployment target for such approaches, following the end-to-end design principle.

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

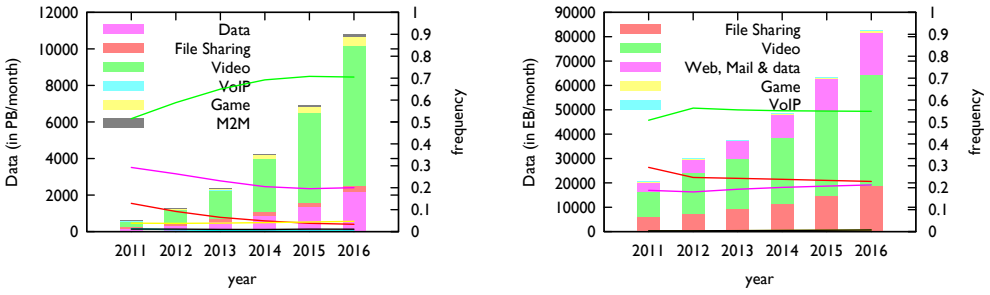
Application	rate	latency	jitter	# connections
web	0	0	0	0
video	0	0	0	0
p2p	0	0	0	0
voip	0	0	0	0
game	0	0	0	0

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

inherent difficulties for evolution. In Section 1.2, we list our contributions 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

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 ideas that formed the subjective condition of the digital revolution of our era, was the concept of computer networking. The initial goal of this concept was to develop a new communication architecture that would allow continuous communication over a redundant network, even when a significant number of vertex 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 [1972b] 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 boost their role, was the radical reduction in cost, size and capabilities of network-enabled personal computers, following Moore's Law model. The low cost factor of personal computers along with the programmable nature of the computer CPU, makes it an elegant platform to develop applications that introduce seamlessly new functionalities. 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 **?**. On one hand, this augmentation in computational devices requires new modes of communication that allow devices to share consistently state, driving a significant development in computer network technologies. A number of network-enabled applications are developed to address these requirements, while new network paradigms are introduced like home networks and hotspots. Existing computer network technologies make this work easy, as they only require a minimal implementation of a protocol and a peer with a forwarding entity in order to establish connectivity with any other device. 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 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 the personal computer paradigm, computer

network are widely adopted as an integral asset for industry. Currently the Internet 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. 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 **??**, 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 don't have access on the properties of the underlying path. The OSI model or the TCP/IP implementations provide semantics for the interfaces but they don't provide any guarantees on the performance properties. Applications on the other hand, have performance requirement which they try to 'enforce' in the deployed environment. As a result, 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 during network congestion.

Network planning for long term periods is also difficult to address. 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 **??** 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. [2007]; 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 should not consuming a significant portion of the computational resources of a node. As a result, the initial security requirements from computer network technologies were minimal. 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 **?**. 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]; **?**]; **?**. 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. We currently have a multitude of protocol 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 protocol define a number of mechanisms that allow protocol designers to develop extensions. Unfortunately, these mechanisms are not guaranteed to be supported across the network, as their support 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

This chapter contains the results of an exhaustive characterisation study of existing SDN technologies. We are trying to understand the capabilities of the SDN paradigm as well as the limitation incurred by current implementation efforts. The work focuses on the functionality of version 1.0 of the OpenFlow protocol, as the sole protocol instatiation of this technology. For this work, we have developed two frameworks: OFLOPS and SDNSIM. OFLOPS is a high precision OpenFlow switch micro-benchmark platform. Using the platform, we develop a set of elementary testing scenario in order to understand the performance limitation of existing OpenFlow switch implementations. SDNSIM is a macro-benchmark platform for OpenFlow architectures. The platform extends the Unikernel abstraction in order to support large scale network simulation and emulation, using the same experiment specification.

In this Chapter, we firsty present the motivations (Section 3.1) and the design overview of the OFLOPS platform (Section 3.2). Further, we select a number of off-the-self OpenFlow implementations (Section 3.3) and present the results of OFLOPS test scenarios over the elemenetary interactions provided by the protocol (Section 3.4). Furthermore, in order to study the results extracted from the oflops platform in a macroscopic level, we present the SDNSIM platform (Section 3.5) and its design (Section 3.7). Finally, we analyse the performance of the SDNSIM implementation and employ it to evaluate network control approaches in a fat-tree topology (Section 3.8) and conclude (Section 3.10).

3.1 OFLOPS introduction

Although the short period since the introduction of the SDN paradigm, many novel control architectures have been proposed Handigol et al. [2009]; Sherwood et al. [2010]; Yu et al. [2010]. The deployment of such architectures to production networks isn't straightforward. In Weissmann and Seetharaman authors describe their experience in deploying the first OpenFlow production networks in Stanford University and point out that the initial deployment was highly under-performing and unreliable when a simple reactive control scheme was deployed. The main source of problems relates to firmware and hardware limitations. In order though to make the transition of SDN and OpenFlow from research to industry , developers need to evaluate thoroughly proposed functionality and provide accurate availability and performance guarantees. In order to address this issue we developed OFLOPS [1], a measurement framework that enables rapid development of use-case tests for both hardware and software OpenFlow switch implementations. To better understand the behavior of the tested OpenFlow implementations, OFLOPS combines measurements from the OpenFlow control channel with data-plane measurements. To ensure sub-millisecond-level accuracy of the measurements, we bundle the OFLOPS software with specialized hardware in the form of the NetFPGA platform[2]. Note that if the tests do not require millisecond-level accuracy, commodity hardware can be used instead of the NetFPGA Arlos and Fiedler [2007].

3.2 OFLOPS design

Measuring OpenFlow switch implementations is a challenging task in terms of characterization accuracy, noise suppression and precision. Performance characterization is not trivial as most OpenFlow-enabled devices provide rich functionality but do not disclose implementation details. In order to understand the performance impact of an experiment, multiple input measurements must be monitored concurrently. Further, current controller designs, like SNA [2010]; Gude et al. [2008a], target production

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

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

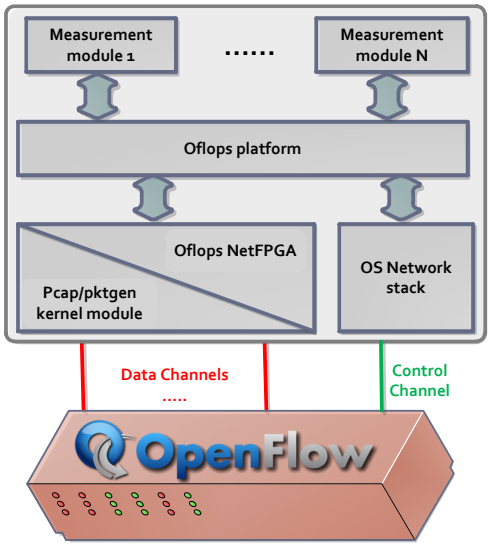


Figure 3.1: OFLOPS design schematic

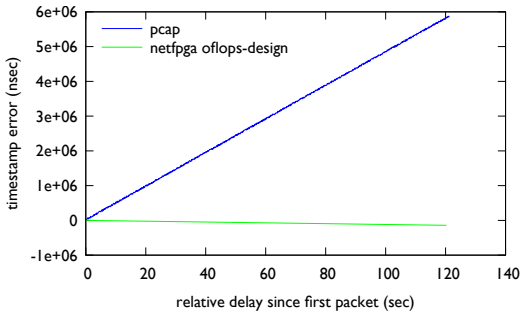


Figure 3.2: Evaluating timestamping precision using a DAG card.

networks and thus are optimized for throughput maximization and programmability, but incur high measurement inaccuracy. Measurement noise suppresion 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.

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

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.

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 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.

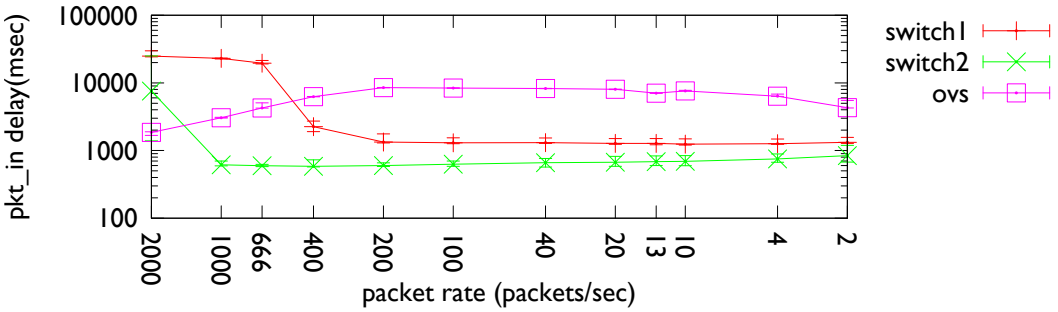
Mod. type	Switch 1			ovs			Switch 2		
	med	sd	loss%	med	sd	loss%	med	sd	loss%
Forward	4	0	0	35	13	0	6	0	0
MAC addr.	4	0	0	35	13	0	302	727	88
IP addr.	3	0	0	36	13	0	302	615	88
IP ToS	3	0	0	36	16	0	6	0	0
L4 port	3	0	0	35	15	0	302	611	88
VLAN pcp	3	0	0	36	20	0	6	0	0
VLAN id	4	0	0	35	17	0	301	610	88
VLAN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

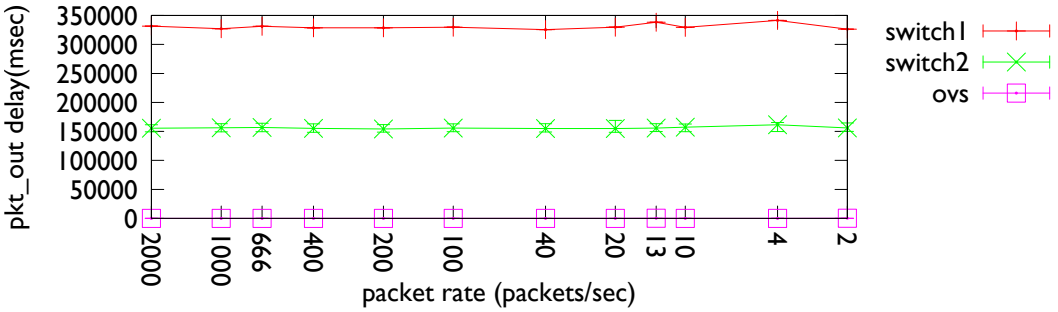
Table 3.2: Time in μs to perform individual packet modifications and packet loss. Processing delay indicates whether the operation is implemented in hardware ($<10\mu$s) or performed by the CPU ($>10\mu$s).

3.4.2 Traffic interception and injection

OpenFlow protocol permits a controller to intercept or inject traffic over the control plane. This functionality allowed in the initial design of the OpenFlow controller to be reactive 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 data rate Kobayashi et al.. This is a direct consequence of the silicon design in current OpenFlow switches, that develop such functionality as an low-frequency exception mechanism. In order to characterise this functionality, we develop a simple experiment using OFLOPS that sends packets at a specific rate and measure the latency of the switch to process them.



(a) Packet in message latency

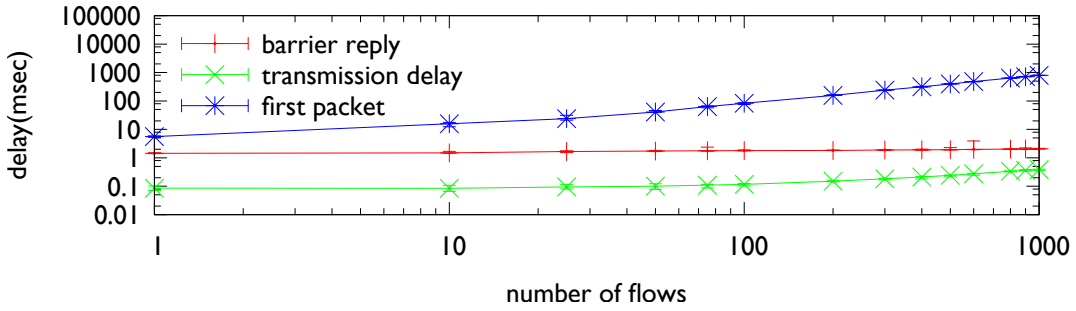


(b) Packet out message latenct

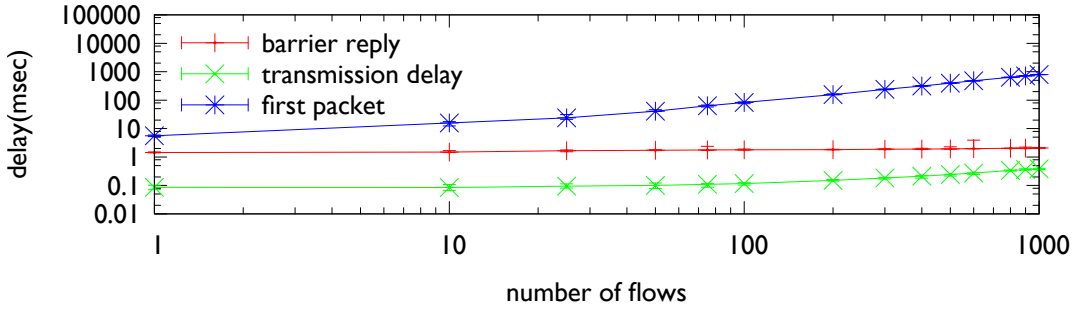
Figure 3.3: Latency to intercept or inject a packet using the OpenFlow protocol

In Figure **?**, we show the latency induced on packets both for Packet_in and Packet_out messages. We omit in this experiment Switch 3 as this functionality maxed out its CPU utilisation and after a few seconds made the switch unresponsive over the control channel. For packet_out messages, the switches rate limit through the tcp connection channel 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 packet important packet at high rates of 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

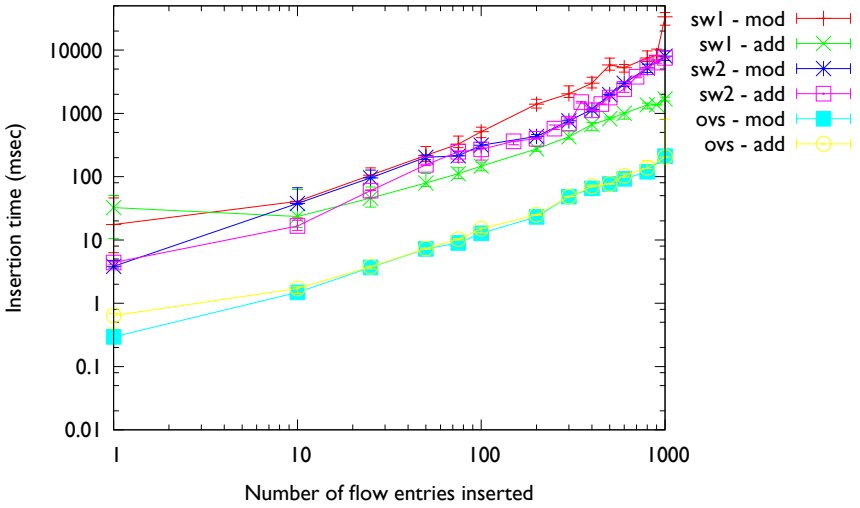


Figure 3.5: Delay of flow insertion and flow modification, as observed from the data plane (log-log scale).

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. Surprisingly, both Switch1 and Switch2 take more time to modify existing flow entries compared to adding new flow entries. For Switch1, this occurs for more than 10 new entries, while for Switch2 this occurs after a few tens of new entries. After discussing this issue with the vendor of Switch2, we came to the following conclusion: as the number of TCAM entries increases, updates become more complex as they typically requires re-ordering of existing entries.

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.

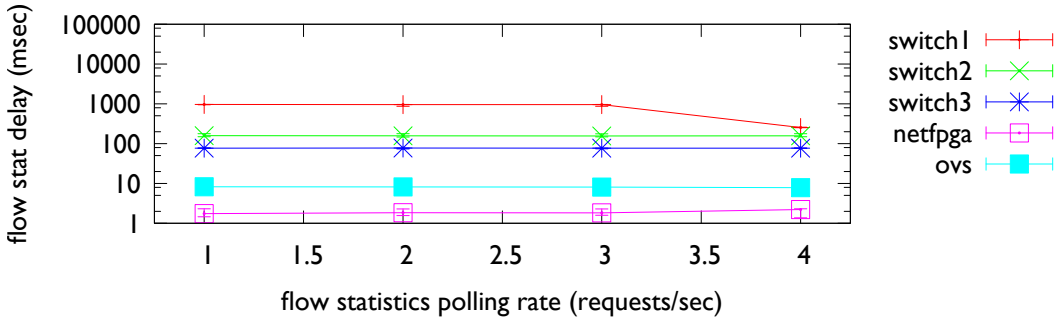
3.4.4 Flow monitoring

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.

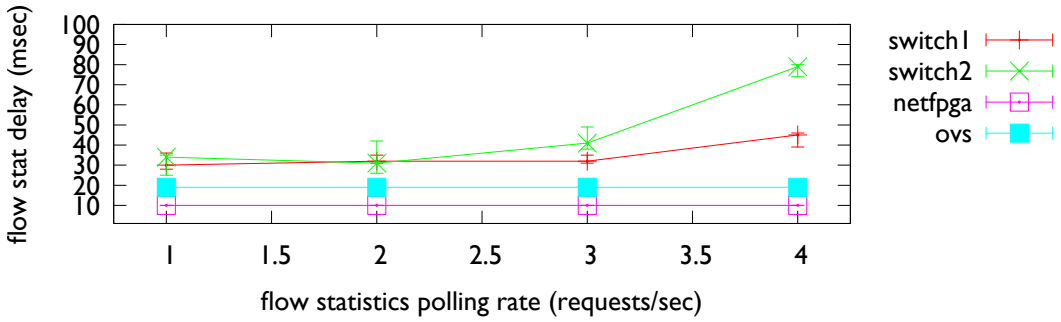
We now test the performance implications of the traffic statistics reporting mechanism of OpenFlow. Using OFLOPS, we install flow entries that match packets sent on the data path. Simultaneously, we start sending flow statistics requests to the switch. Throughout the experiment we record: the delay getting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



(a) Reply time.



(b) CPU utilization.

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

low polling rates the switch splits its answer across multiple TCP segments: each segment containing statistics for a single flow. As the probing rate increases, the switch will aggregate multiple flows into a single segment. This suggests that independent queuing mechanisms are used for handling flow statistics requests. Finally, neither software nor NetFPGA switches see an impact of the flow-stats rate on their CPU, thanks to their significantly more powerful PC CPUs (Table 3.1).

3.4.5 OpenFlow command interaction

An advanced feature of the OpenFlow protocol is its ability to provide 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.

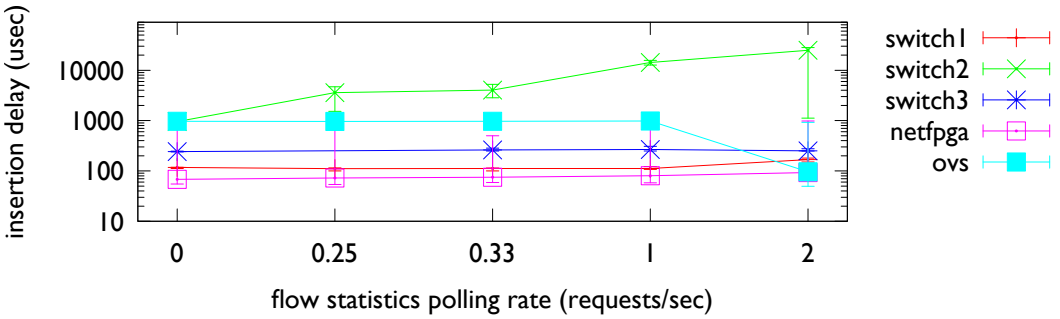


Figure 3.7: Delay when updating flow table while the controller polls for statistics.

[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 SDNSIM introduction

In the SDN paradigm, backward-compatible evolvability of computer networks is achieved through the distribution of control to external programmable units. The protocols provides all required mechanisms in order to allow a remote entity to get sufficient forward plane feedback and control the forwarding process at a very fine level. So far the trend in OpenFlow design is to aggregate control in a single control unit, in order to have a single point of control in the network. This aggregation permits on one hand to achieve higher optimality in forwarding policy, while on the other hand the logic can be developed in richer programming environments, than the embedded systems usually found in current network devices.

The ability to distribute control over multiple functional units, in conjunction with the complex nature of network stacks, as well as the diverse behaviour of OpenFlow switch and controlling platforms, reduces the ability of developers to predict the behaviour of an SDN design. In order to reason on correctness, developers must build realistic small scale experiments, and larger scale experiments in order to reason on performance. In the related literature on network evaluation there have been two main experimental approaches: *realistic testbed* and *simulation*.

In the realistic experimentation approach, scientists try to reconstruct the exact properties of the deployment environment. This approach provides an optimal measurement environment the provides full control of the parameter of the experiment, but has a significant overhead in resources and configuration time. Experimenting with a real datacenter would require a high number of machines that implement full functionality of the target environment, large interconnection planning and careful metrication and analysis of the resulting system. These requirements scale badly as the number of network hosts increases. In order to reduce resource costs, the research community has developed shared testbeds *add some pointers.* . Such testbeds though limit the capability of users to control measurement noise and network topologies.

In the simulation approach, researchers replace some part of the functionality of the system by a simpler model. The goals of such an approach is to reduce the complexity of the experiment, and achieve scalability for large network sizes. This approach has two main drawbacks. Firstly, the fidelity of the results depends extensively on the reality of the assumptions of the model. Secondly, in order to simulate appropri-

ately the network, the users usually need to rewrite the logic of their experiments in order to fit the abstraction of the underlying models. For example, in order to simulate OpenFlow-based network currently there is no off-the-self simulation platform to support the protocol. An OpenFlow architect is required to reimplement its OpenFlow logic in order to experiment with architectural designs.

SDNSIM [1] is a novel framework, that bridges the two domains of experimentation and provides an OpenFlow specific development environment. 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 required functionality of their network design and at the compilation step select the target experiment type. SDNSIM provides two target options: a simulation target, that runs the Mirage code on top of the ns-3 simulation environment, and a realistic target, that build and deploys each node of the simulation as a DomU VM and configuration resource allocation using xen's API.

3.6 Mirage Library OS

Cloud computing has revolutionized the way the business use IT infrastructures as well as the way we develop distributed computing applications. The abstraction is straightforward. A third party entity takes responsibility to maintain a datacenter. This infrastructure, is partitioned into smaller virtual computational units which clients can rent in order to run their applications. The elegance of this model is based on simplicity of the abstraction that the cloud provider provides to the user and the ability to port existing services running on a personal computer or a server to the cloud platform.

Although the simplicity of the exposed abstraction, the cloud architecture consist of a complex set of processing layers. A single application VM would include: i) the virtualization runtime layer, ii) the guest OS kernel layer, iii) A language runtime layer (POSIX, JVM etc.) iv) user-space thread layer. This layer complexity, although it provides excellent backwards compatibility for existing datacenter applications, it makes the process of optimisation, debugging as well as security difficult, while there

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

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

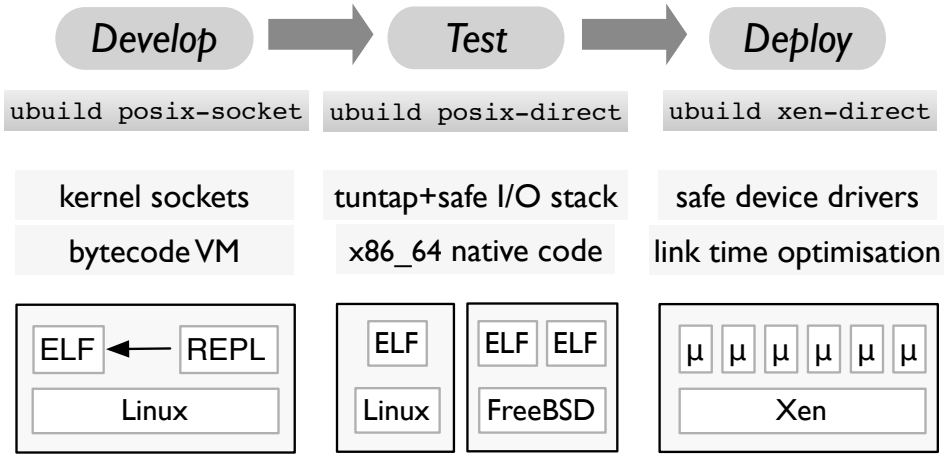


Figure 3.8: Specialising a Mirageapplication through recompilation alone, from interactive UNIX Read-Eval-Print Loop, to reduce dependency on the host kernel, and finally a unikernel VM.

is a significant overlap on the functionality provided by each layer.

Mirage is a clean-slate application synthesis framework that allows users to compile application to a single bootable VM image. Mirage relies on the idea of library OS; functionality is separated into logical modules which integrates in a resulting binary, only if needed by the code. As a result, Mirage is able to generate small size VM images which additionally are fast to boot. Although, OCaml is a functional language, it is able to generate highly performand binary code and application specific microbenchmarks has shown performance to be compare to c code implementations. In order to minimize processing layers, IO operations are coded on top of the Net-Front device abstraction provided by Xen, while the OS is using extensively zero-copy mechanisms and exposes directly the pages of the shared memory ring to application space, over a simple abstraction that enforces memory boundaries.

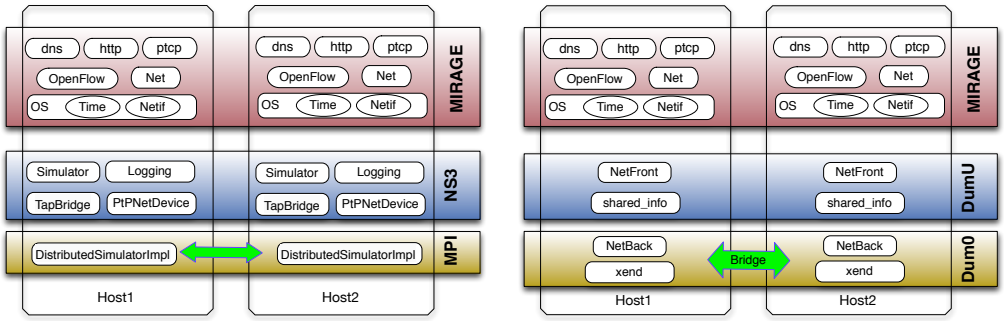
Mirage executes OCaml code over a specialised language runtime modied in two key areas: memory management and concurrency. Because of the single process nature of Mirage images, the OS uses a single address space, separated between the text and data section of the program and the runtime heap. Because the program code is immutable during runtime, Mirage can lock write access to executable memory space, thus mitigating buffer overflow attacks. Additionally, this mechanism removes the re-

quirement of Address Space Randomization (ASR), which simplifies and makes more efficient memory management. To provide concurrency beyond Xen IO polling function, Mirage integrates the Lwt cooperative threading library **?**. This internally evaluates 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.

In order to provide basic system programming capabilities, Mirage OS provides two core modules, named Net and OS, that implement and expose the required functionality for networking and device management, respectively. The API is similar to the API provided by the Unix model from OCaml; developers can modify their code and compile existing applications to Mirage VMs. Due to the simplicity of the OS and Net modules, Mirage provides the ability 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 the BareboneOS, an assembly written minimum OS. The diverse set of deployment backends, provides to developers an environment to test and optimize code gradually. 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 evaliation, and finally they can synthesize the resulting deployable Xen Image 3.8.

3.7 SDNSIM design

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<topology module="Simple_tcp_test" backend="ns3-direct"
```



(a) NS3 (b) Xen

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

```
  duration="30">
 <modules>
  <library>lwt</library>
  <library>lwt.syntax</library>
  <library>cstruct</library>
  <library>cstruct.syntax</library>
  <library>mirage</library>
  <library>mirage-net</library>
  <library>pttcp</library>
 </modules>
 <node name="host1" main="host_inner">
  <param>1</param>
 </node>
 <node name="host2" main="host_inner">
  <param>2</param>
 </node>
 <link src="host1" dst="host2" delay="10" rate="100"
  queue_size="100" pcap="false"/>
</topology>
```

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

For the user perspective, the core of SDNSIM consists on a single executable that works as one more OCaml build system. Developers write their OCaml code in order to define the functionality of network nodes, and use a single xml file in

order to describe the network topology and match functionality to network hosts. 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 can be seen in Figure 3.9. At the top layer of the host is the application logic of the host. This layer is defined 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 Mirageplatform, namely dns, http, ssh and OpenFlow. Additionally, we have re-implemented in OCaml the pttcp tcp test tool functionality **??**, in order to allow model driven TCP flow 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 this platforms aims 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 interaction, a different strategy has been followed for each backend.

3.7.1 Xen

In order to run a Mirage Xen Image, the project provides a simple PV Boot mechanism. The mechanism initializes a VM with a single virtual CPU, loads the Xen event channel and jumps to a main function of the VM. The main function firstly enumerates IO

devices and notifies the application and then proceeds to the main threading scheduling logic. 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 calculated 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.

In order to integrate the clock of a mirage instance with the global clock of the Xen platform, each time request is mapped to the time counter provided by Xen shared_info struct. In respect to the network functionality of Mirage, network packets are mapped as IO events on the network device of the NetFront driver. Using this simple logic, users can develop highly functional appliances which can be deployed as small factor image over the Xen platform.

Further, in order to express network topologies over the Xen platform, SDNSIM takes advantage of the Xen API provided by the latest version of the Xen platform. The API provides the ability to fully control functionality and resource allocation for each VM running on a Xen domain. SDNSIM, using the Xen API, is able to implement network topologies through vif bridging between hosts, while link rate and propagation delay can be encoded in the vif definition. When SDNSIM is instructed to run a real-time experiment, it firstly compiles all required VM images, creates the appropriate host definition and network topologies and when the configuration step is completed it can fire the experiment.

3.7.2 NS3

NS3 is a framework for discrete time packet driven network simulation. It builds around a basic event-driven simulation engine and provides a set of libraries that implement 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.

Since the way that the NS3 platform is event driven and the scheduling mechanism is hidden in the Simulation engine, we followed a different approach in order to map Miragefunctionality. Firstly, in order to bridge the simulation clock with the Mirage host time, we have mapped the gettimeofday functionality of the OS with the global Simulator clock of the NS3 platform. Further, in order to provide accurate time events in the Mirageplatform we modified the main mechanism for time events in the platform, the sleep function. Each sleep call is blocked and scheduled as an NS3 timed event. The thread is resumed only when the simulator fires the respective timed event. Finally, in order to make sure that any sleeping thread is not paused for a long period, the OS schedules *i*dle timed events that check for resume any yielded threads.

The networking functionality uses the simple link abstraction of a *P*ointToPoint channel. This model simulates a ppp link over a lossless medium, a valid model for the full duplex non-shared medium of current network datacenters. Traffic transmission, uses a single packet queue per network device shared between the Miragelayer and the NS3 simulator engine. The only modification we make on the design of the ns3 link is to modify the network queue functionality of the network device abstraction. The queue implement a simple bidirectional feedback mechanism in order to avoid any buffer overflows, as well as avoid queue polling to check if the queue has space for new packets. For packet receipt, the NS3 device allows registering receipt callback to each virtual network device. The packet demuxing callback implements logic to forward packets to the appropriate listening thread.

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-threaded programming. This simplification provides more predictable performance for the program, as the garbage collector doesn't have to manage threads synchronisation, while scalability can be achieved through multiprocess programs that synchronise through other IPC mechanisms. In order to make SDNSIM scalable for large network sizes we followed a similar approach. For the simulation engine we use DistributedSimulationImpl class, which uses message passing techniques in order to coordinate multiple simulation instances 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 microbenchmarks in order to evaluate the performance of the OpenFlow protocol functionality as well as the performance of the NS-3 binding. In **?**, there is an exhaustive analysis of the performance of the Mirageplatform, which we omit from this section. In Section **??**, we use two off-the-self OpenFlow benchamarking 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 MirageController

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 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 **?** 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

[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.

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 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.

3.8.2 MirageSwitch

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

[1] http://openvswitch.org

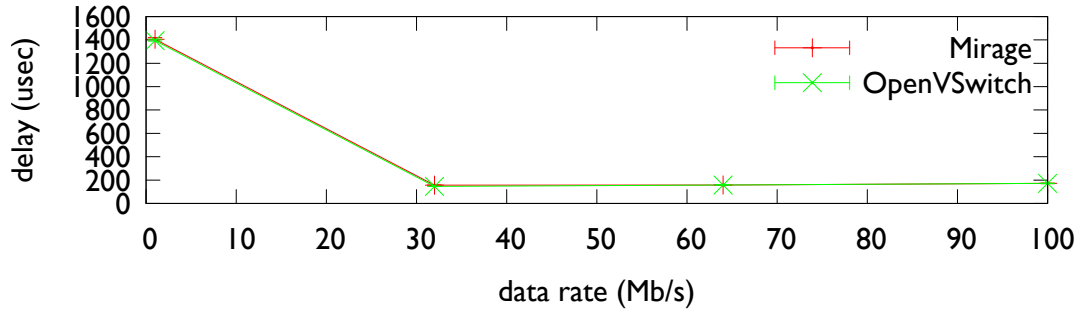


Figure 3.10: Min/max/median delay switching 100 byte packets when running the Mirage switch and Open vSwitch kernel module as domU virtual machines.

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 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.

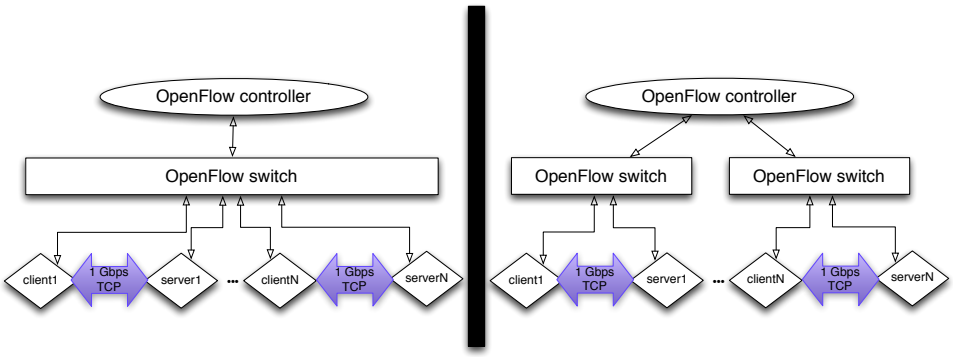


Figure 3.11: Topology of two basic simulation scenarios for the SDNsim platform

3.8.3 NS-3 performance

In order to test the scaling properties of the NS3 backend we perform a simple topology experiment (Figure 3.11) with variable number of hosts. The hosts are generated in pairs and are programmed to generate steady-state full-rate TCP traffic between them. We use two variations of the topology: A centralised example where all hosts are connected to a single switch, and a localised example, where hosts are distributed between two switches and traffic is local to the switch.Each switch is connected to an OpenFlow controller that implements a learning switch. The 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 the hosts of the simulation create small autonomous partition. In the centralised example, the OpenFlow switch is the bottleneck of the simulation, as it has to process sequentially all network events. In the distributed example, the network event processing is distributed between the two switches of the topology, thus achieving better simulation parallelization.

	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.

3.9 Security Tradeoffs on Datacenter Network Micro-control

3.10 Summary and Conclusions

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 **??** we present the outcome of a number of studies on the connectivity properties and the traffic mix of the environmeny, and in Subsection **??** we present the finding of home network ethnographic studies that hightlight the social aspect of the technology.

4.1.1 Home Network Technologies

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. 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 the local network properties as well as the nature of the Internet related 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 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

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

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 to the home network (e.g. smartphones, tablets, game consoles, etc.), which require from any designed solution to be as much as possible open to network 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 ssids discovered on the end-host during the measurement, while approx 1/3 of the Wifi networks overlap on the channel with neighbouring networks. These observations also 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 express different traffic mix. Specifically, users tend to use more network filesystems and p2p applications, while a large protion of the traffic remains local and can only be observed from within the network.

Netalyzer **?**, 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. In terms of network performance, recently a number of govermental organisations have started to develop monitoring mechanisms in order to understand

the service that ISP provide to users in a national level **??**. Currently, there are two dominant link technologies available for residential networks, ADSL and cable. There are also other connectivity mechanism (3G, fiber), which though are less popular and restricted to specific countries. Work on this field uses either packet traces from the access network, or active probes to a subset of the network. One of the first analysis on this scope can be found in **??**, 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 can be highly variable. In Sundaresan and de Donato [2011] authors follow a different approach in measuring the network link and integrate various measurement tool 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 the Internet traffic mix of home networks, a number of studies on network data from ISP has showed that user trends evolve over time and have changed significantly in the recent years, following the improvement in Internet services. In Cho et al. [2006] authors describe that during the time of their analysis within Japan, there is a significant utilization p2p applications. More recent studies Maier et al. [2009] point out that users in Germany have moved to web applications and a large portion of the Internet traffic is currently 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 Exploring Home Network Ethnography

In the space of home networking a number of ethnographic studies have been presented that try to understand the social aspect of the network. Most studies have performed using user interviews.

user engagement with network and required functionality

expressing user collaboration on the social aspect. how user share their network?

Mismatch of user perception to real properties of the network. e.g. what are the limits of the local network?

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

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 technology and, as this is not a professional activity, rarely motivated to become expert. A wide range of devices connect to the home network, including desktop PCs, games consoles, and a variety of mobile devices ranging from smartphones to digital cameras.

Not only do these devices vary in capability, they are often owned and controlled by different household members.

To illustrate the situation we are addressing, consider the following two example scenarios, drawn from situations that emerged from our fieldwork to date, reported in more detail elsewhere Brundell et al. [2011]:

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.*

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. For example, we may need to explore architectures that sacrifice scalability in favor of installability, evolvability, and maintainability.

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 scalability to the same degree as the core network;
- doing so allows us to monitor traffic in a way that is more meaningful for users; and
- we can apply per-flow queueing mechanisms to control bandwidth consumption, commonly requested by users.

4.3 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

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

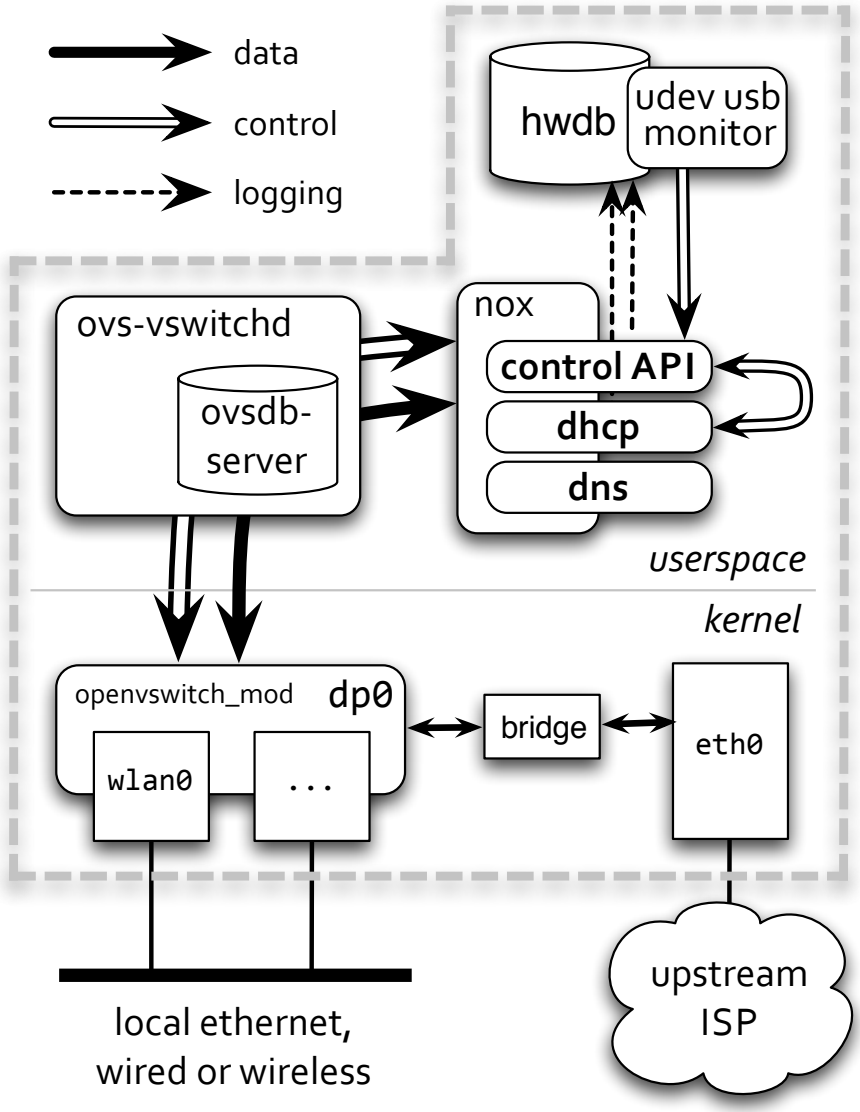


Figure 4.1: Home router architecture. Open vSwitch (*ovs**) and NOX manage the wireless interface. Three NOX modules provide a web services control API, a DHCP server with custom address allocation and lease management, and a DNS interceptor, all logging to the Homework Database (*hwdb*) (§4.4).

traffic, plus the Homework Database Sventek et al. [2011] providing an integrated network monitoring facility. The proposed setup is similar to the standard ISP-provided home router.

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

4.3.1 OpenFlow, Open vSwitch & NOX

OpenFlow is a switching standard McKeown et al. providing an open protocol for distributed control of the forwarding tables contained within Ethernet switches in a network. An OpenFlow *switch* has three parts: a *datapath*, a *secure channel* connecting to a controller, and the *OpenFlow protocol* the controller uses to talk to the switch.

Each datapath applies actions to flows arising on a physical interface, where *flow* is defined as a tuple of the primary packet header fields plus the physical port on which the flow is visible. Flow definition allows wildcarding of fields and specifically permits netmasks for IP addresses. Each flow can have a number of primitive actions applied; actions defined in the protocol permit full control over forwarding as well as modification of all fields of the flow tuple. The net effect is that applications can manage and control traffic according to their own definition of a network flow. Flow entries are installed by the controller when the switch notifies the controller of arrival of a packet from a new flow.

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.

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

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.

Our router provides flow-level control and management of traffic via a single OpenFlow 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

[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.

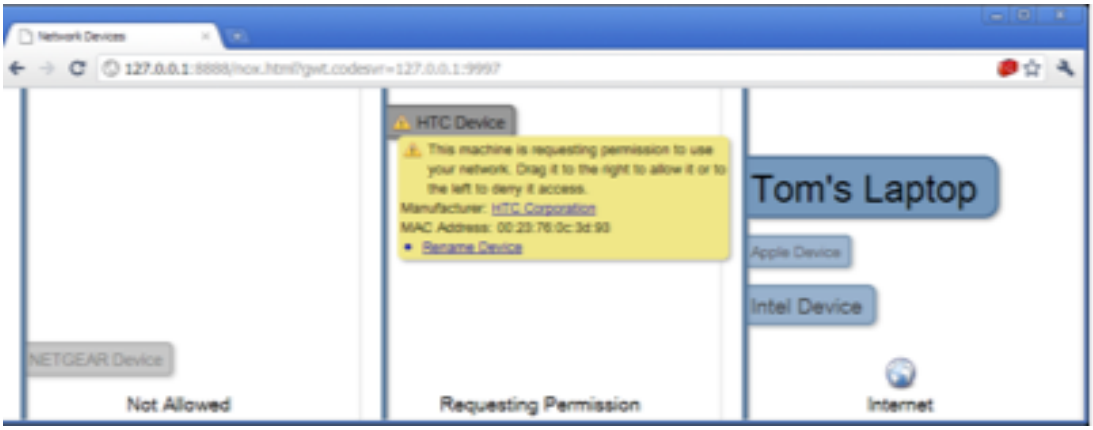


Figure 4.2: The *Guest Board* control panel, showing an HTC device requesting connectivity.

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 events since a timestamp or during an interval defined by two timestamps. *hwdb* is active as applications may register interest in *future* behaviour patterns, triggering notification when such a pattern is detected by the database. The work described in this paper makes use of three tables: *Flows*, accounting traffic to each 5-tuple flow; *Links*, monitoring link-layer performance; and *Leases*, recording mappings assigned via DHCP.

4.3.3 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. The interface runs on a dedicated touch screen in the home and 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.

- Device bandwidth use is proportional to its label size.
- Wireless Ethernet retransmissions show as red highlights on the device's label.

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: 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. While this example describes a central control point in the home, 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. is that traffic between devices in the 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

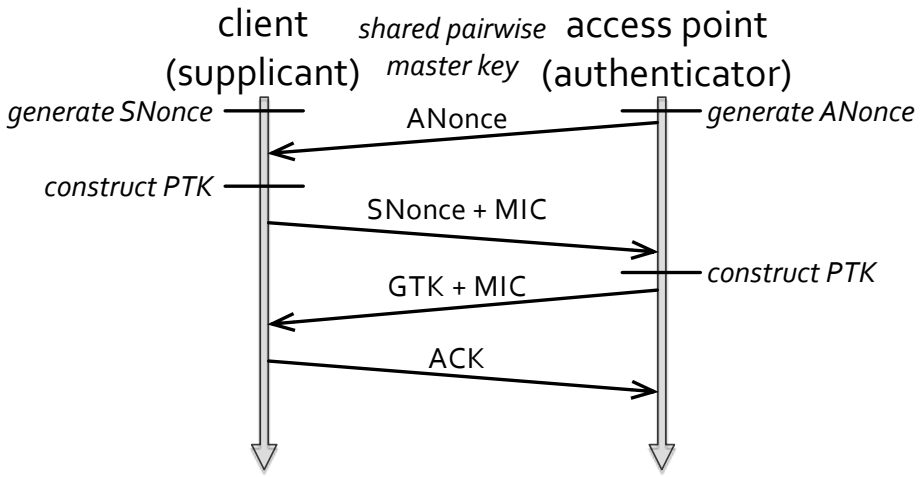


Figure 4.3: 802.11i handshake, part of the association process. Note that MIC (Message Integrity Code) is an alternate term for MAC, used in such contexts to avoid confusion with Media Access Control.
with 10 devices, gives a median per-host service time of 0.7s.

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.

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]

[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

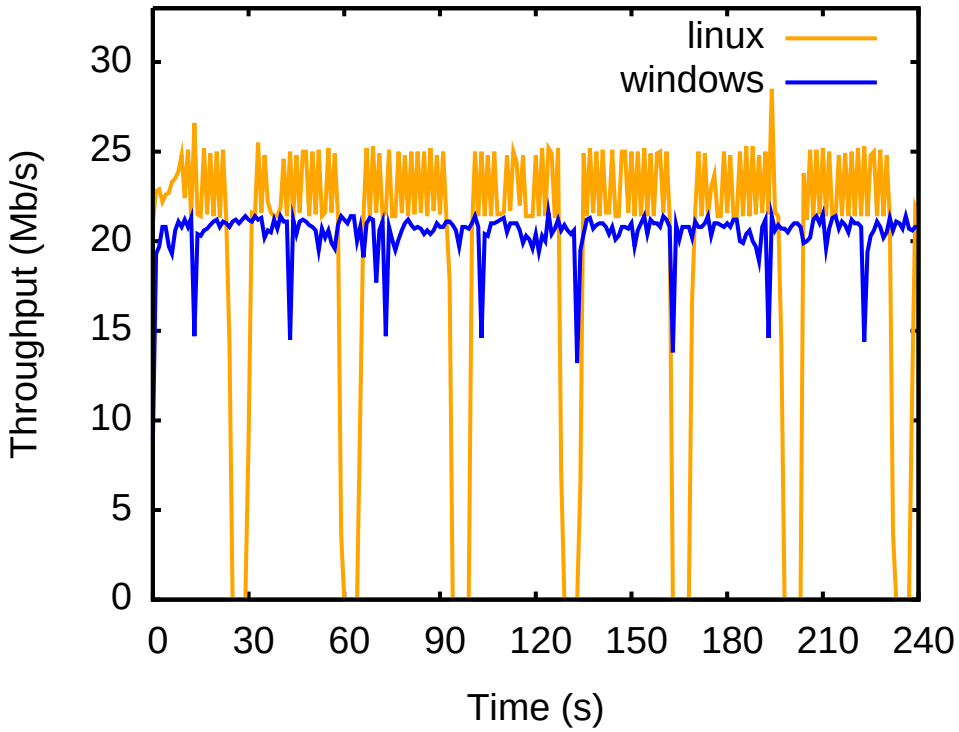


Figure 4.4: Affect on TCP throughput from rekeying every 30s for Linux 2.6.35 using a Broadcom card with the *athk9* module; and Windows 7 using a proprietary Intel driver and card.

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 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

we do not consider it further.

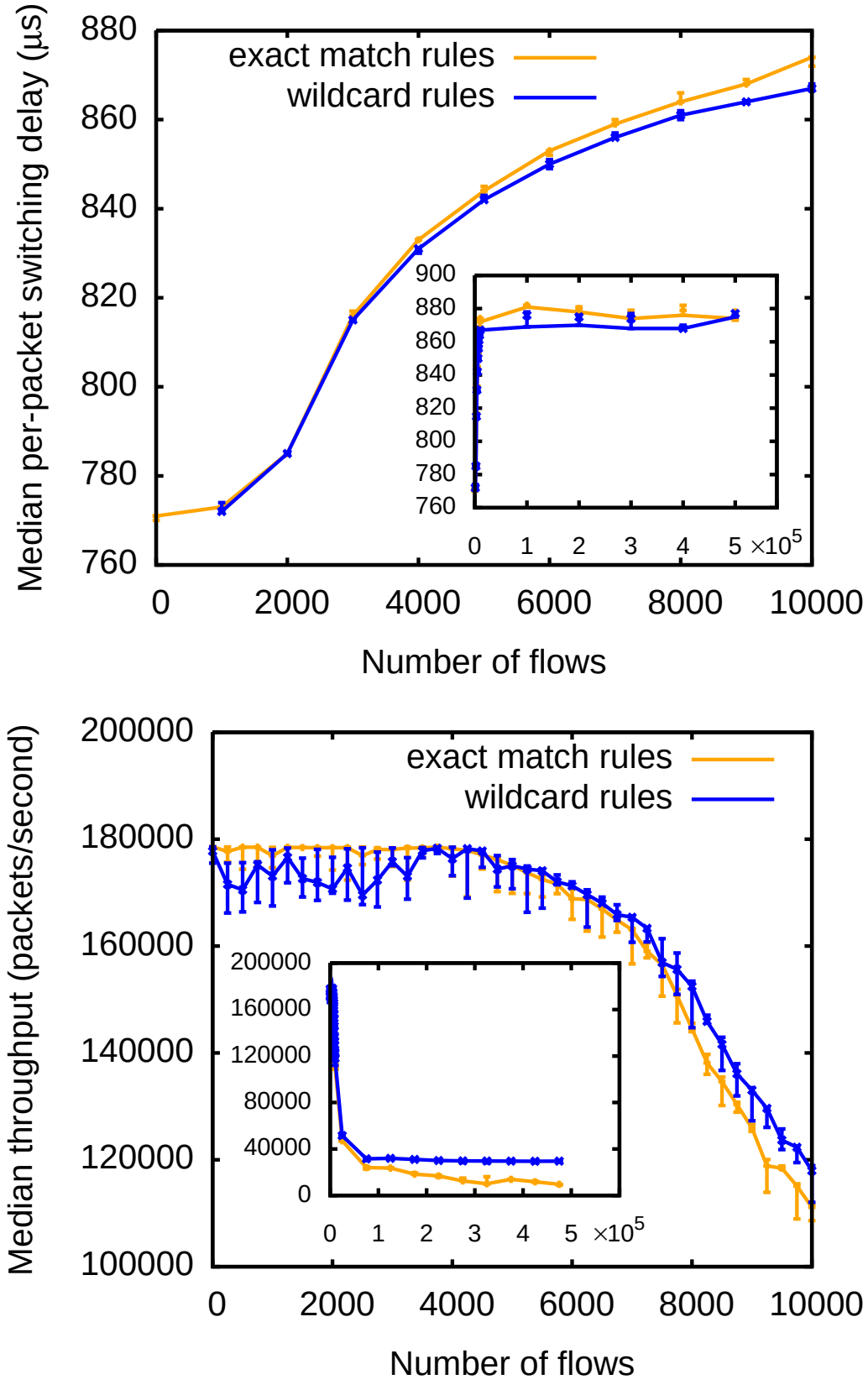


Figure 4.5: Switching performance of Open vSwitch component of our home router showing increasing per-packet latency (LHS) and decreasing packet throughput (RHS) with the number of flows. The inset graph extends the x-axis from 10,000 to 500,000.

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 As a compromise, when a device is blacklisted, all of its traffic and subsequent rekeying exchanges are blocked. The device will be able to receive only broadcast traffic in the interim due, to the use of the GTK for such frames, until the AP initiate the negotiation of a new key.

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. Extending this fine-grained control requires more accurate identification of traffic to application, particularly for more complex network uses such as BitTorrent and Skype, and is a problem we are investigating in ongoing work.

4.4.3 Forwarding

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

Non-IP traffic. The controller acts as a proxy ARP server, responding to ARP requests from clients. Misbehaving devices are blacklisted via a rule that drops their EAPOL Aboba et al. [2004] traffic thus preventing session keys negotiation. Finally, other non-IP 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 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) [2] for the throughput tests, selecting

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

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

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 Open vSwitch can handle orders of magnitude more rules than required by any reasonable home deployment. Nonetheless, to protect against possible denial-of-service attacks on the flow tables, whether 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: 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

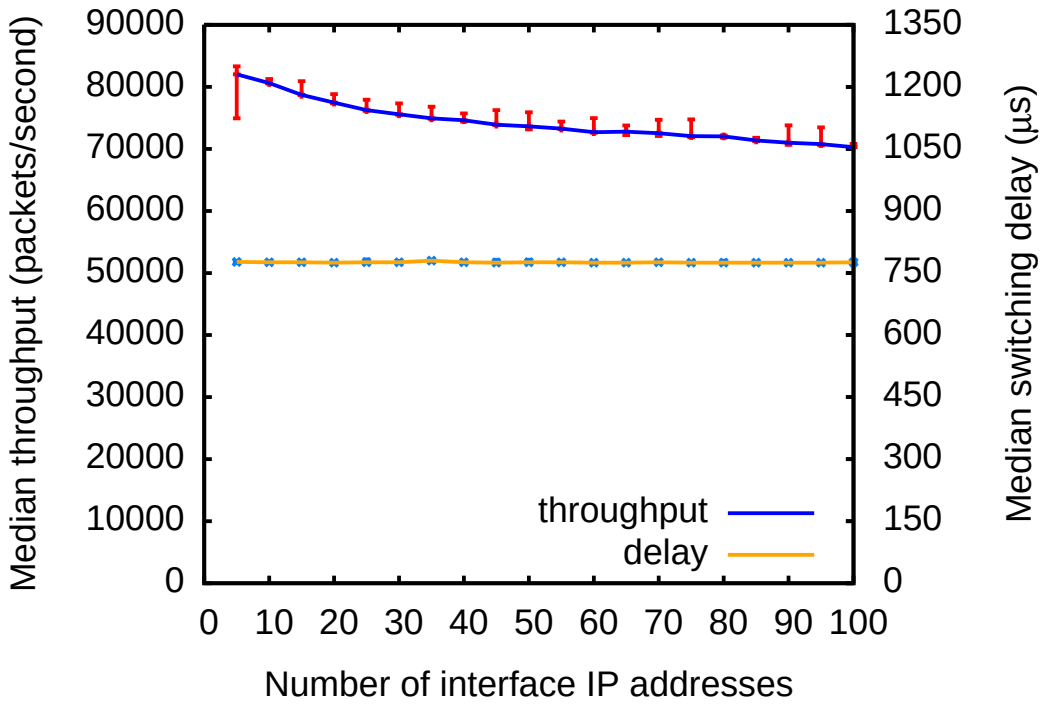


Figure 4.6: Switching performance of Linux network stack under our address allocation policy. Throughput (left axis) shows a small linear decrease while switching delay (right axis) remains approximately constant as the number of addresses allocated to the interface increases.

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.

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

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

Bonjour and Avahi both rely on periodic transmission of multicast DNS replies advertising device capabilities via TXT records. SSDP is similar, but built around 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 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

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

of the total traffic; data from a limited initial deployment (about one month in two homes) suggests that broadcast and multicast traffic combined accounts for less that 0.1% (packets and bytes) in both homes.

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

Addition of IPv6 support affects the network layer only, requiring consideration of routing, translation between network and link layers, and address 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. Similarly, supporting translation to lower layer addresses equates to supporting ICMPv6 Neighbour Solicitation messages which perform equivalent function to ARP.

Address allocation is slightly more complex but still straightforward. IPv6 provides two address allocation mechanisms: *stateless* and *stateful*. The first allows a host to negotiate directly with the router using ICMPv6 Router Solicitation and Advertisement packets to obtain network details, IP netmask and MAC address. Unfortunately this process requires that the router advertises a 64 bit netmask, of which current plans allocate only one per household, with the result that all hosts would end up on the same subnet. The second builds on DHCPv6 where addresses are allocated from a central entity and may have arbitrary prefix length. This would enable our router to function in much the same manner as currently, although it would need to support the ICMPv6 Router Advertisement message in order that hosts could discover it as the router.

[1] OpenFlow aims to provide support in its 1.2 release of the protocol; NOX currently has no support for IPv6; and Open vSwitch only supports IPv6 as an application specific extension.

4.5 User-Driven Resource Management

4.5.1 Helping thy neighbour: Enabling ISP - User collaboration

4.5.2 Architecture

4.5.3 Evaluation

4.6 Conclusions

Chapter 5

Scalable User-centric cloud networking

In this chapter we explore application of the SDN technology in personal cloud computing applications. In the recent years, the development in cloud computing applications provided a number of popular Internet-wide services that allow users to interconnect at the application layer and share information. Through this approach users bypass the restrictions imposed in the Internet by deployed middleboxes, at the cost though of exposing their information to third party cloud service providers and reduced performance and efficiency.

Using SDN technologies, we design an architecture that allows users to interconnect their devices with minimum interactions with the cloud infrastructure.The propose architecture deploy an openflow-enabled bridge and a local controller on each device. Each controller by default forwards packets as normal on the local network. The forwarding logic, though, is augmenting through a distributed coordination protocol which permits nodes to negotiate possible connection opportunities and establish ad-hoc tunnels, enabling as a result an Internet-wide distributed control mechanism. At the core of our design, we tranform the naming service host abstraction. Each device acquires a global domain name, while each name resolution triggers a connection engine, that tries to find the best possible bidirectional channel between the two nodes. The naming service uses the established DNSSEC extension, providing a fully authenticated and secure control mechanism among Signpostand applications. Further,

the distributed nature of the naming hierarchy in the Internet permits seamless control distribution.

In order to understand the impact of the proposed architecture we develop a strawman implementation, named *Signpost*. Signpost implements the core of the control logic of the proposed architecture. Additionally, it integrades a number of network tactics of established tunneling and notification mechanisms. Currently Signpostprovides integration of the main architecture with SSH, OpenVPN, TOR, Privoxy, Multicast-DNS and Nat punching.

In the chapter we present in Section 5.1 the motivation for this work, followed then by the key observations for our design in section 5.2. Following the results of our observations, we present in section 5.3 the architecture of our proposed proposed strawman implementation of the controller and the details of the tactics. Finally, in Section 5.4 we present a number of micro-benchmark tests for our system and conclude in Section5.5.

5.1 Introduction

Present the gap in network understanding between the clients and the ISP and present a tool that allows to bridge it.

5.2 Enabling edge user-driven connectivity

5.3 Signpost Architecture

5.4 Evaluation

5.5 Conclusions

Chapter 6

My Conclusions ...

Here I put my conclusions ...

References

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

Openflow switch specification (version 1.0.0). www.openflow.org/documents/openflow-spec-v1.0.0.pdf, December 2009. 16, 17, 21

The snac openflow controller, 2010. http://www.openflow.org/wp/snac/. 13

B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, H. Levkowetz, and Ed. Extensible Authentication Protocol (EAP). RFC 3748, IETF, June 2004. 56

A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15(2), June 2005. 49

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

R. Atkinson. RFC 1825: Security architecture for the Internet Protocol, August 1995.

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

S Bauer, R Beverly, and A Berger. Measuring the state of ECN readiness in servers, clients, and routers. pages 171–180, 2011. 8

A. Bhushan, B. Braden, W. Crowther, E. Harslem, J. Heafner, A. McKenzie, J. Melvin, B. Sundberg, D. Watson, and J. White. RFC 172: The file transfer protocol, June 1971a.

A. Bhushan, R. Braden, W. Crowther, E. Harslem, J. Heafner, A. McKenzie, J. Melvin, B. Sundberg, D. Watson, and J. White. RFC 171: The Data Transfer Protocol, June 1971b.

A. Bhushan, R. Braden, W. Crowther, E. Harslem, J. F. Heafner, A. M. McKenzie, J. T. Melvin, R. L. Sundberg, R. W. Watson, and J. E. White. RFC 265: The File Transfer Protocol, December 1971c.

A. Bhushan, B. Braden, W. Crowther, E. Harslem, J. Heafner, A. McKenzie, B. Sundberg, D. Watson, and J. White. RFC 264: The data transfer protocol, January 1972.

A. K. Bhushan. RFC 114: File transfer protocol, April 1971.

A. K. Bhushan. RFC 294: The use of "set data type" transaction in file transfer protocol, January 1972a.

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

A. K. Bhushan. RFC 385: Comments on the file transfer protocol, August 1972c.

A. K. Bhushan. RFC 414: File transfer protocol (FTP) status and further comments, December 1972d.

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

R. T. Braden. RFC 238: Comments on DTP and FTP proposals, September 1971.

H. Brodie. RFC 269: Some experience with file transfer, December 1971.

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. 45

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

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, 46

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

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

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 2007. 5

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

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

R.E. Grinter and W.K. Edwards. The work to make the home network work. In *Proc. ECSCW*, 2005. 44

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. 13

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, 25

E. Harslem and J. F. Heafner. RFC 141: Comments on RFC 114: A file transfer protocol, April 1971.

E. Harslem and R. Stoughton. RFC 225: Rand/UCSB network graphics experiment, September 1971.

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

M. Horowitz and S. Lunt. RFC 2228: FTP security extensions, October 1997.

J Huang, Q Xu, B Tiwana, Z M Mao, M Zhang, and P Bahl. Anatomizing application performance differences on smartphones. pages 165–178, 2010. 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

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. 19

M. Krilanovich. RFC 399: SMFS login and logout, September 1972a.

M. Krilanovich. RFC 431: Update on SMFS login and logout, December 1972b.

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

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

N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling innovation in college networks. whitepaper, http://www.openflowswitch.org/wp/documents/. 48

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

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

T. H. Myer and D. A. Henderson. RFC 680: Message transmission protocol, April 1975.

Jad Naous, David Erickson, G. Adam Covington, Guido Appenzeller, and Nick McKeown. Implementing an openflow switch on the netfpga platform. In *ANCS*, 2008. 16

N. Neigus. RFC 542: File transfer protocol, July 1973.

A. G. Nemeth. RFC 43: Proposed meeting, April 1970.

R. Olsson. *pktgen*, the linux packet generator. In *Proc. Linux Symposium*, 2005a. 57

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

J. Postel. RFC 765: File transfer protocol specification, June 1980.

J. Postel and J. K. Reynolds. RFC 959: File transfer protocol, October 1985.

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

T. Rodden and A. Crabtree. Domestic routines and design for the home. *Computer Supported Cooperative Work: The Journal of Collaborative Computing*, 2004. 44

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

T. Rodden, A. Crabtree, T. Hemmings, B. Koleva, J. Hunble, K.-P. Akesson, and P. Hansson. *Assembling connected cooperative residential domains*, pages 120–142. Springer, 2007. In *The Disappearing Computer: Interaction Design, System Infrastructures and Applications for Smart Environments* (eds. Streitz, N., Kameas, A., and Mavrommati, I.). 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. 21

E. Shehan and W.K. Edwards. Home networking and HCI: What hath God wrought? In *Proc. ACM CHI*, 2007. 44

E. Shehan-Poole, M. Chetty, W.K. Edwards, and R.E. Grinter. Designing interactive home network maintenance tools. In *Proc. ACM DIS*, 2008. 44

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

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.-Y. Sung, L. Guo, R.E. Grinter, and H.I. Christensen. My roomba is rambo: Intimate home appliances. In *Proc. UbiComp*, 2007. 44

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, 49

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

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

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

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

J. E. White. RFC 74: Specifications for network use of the UCSB on-line system, October 1970.

J. E. White. RFC 105: Network specifications for remote job entry and remote job output retrieval at UCSB, March 1971a.

J. E. White. RFC 122: Network specifications for UCSB's simple-minded file system, April 1971b.

J. E. White. RFC 217: Specifications changes for OLS, RJE/RJOR, and SMFS, September 1971c.

Kok-Kiong Yap, Masayoshi Kobayashi, David Underhill, Srinivasan Seetharaman, Peyman Kazemian, and Nick McKeown. The stanford openroads deployment. In *Proceedings of ACM WINTECH*, 2009. 25

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