Flexible Software Defined Network

Charalampos Rotsos

Computer Laboratory

University of Cambridge

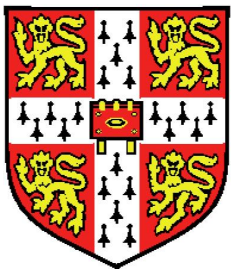
A thesis submitted for the degree of

Doctor of Philosophy

Yet to be decided

Abstract

Flexible Software Defined Network



Charalampos Rotsos

Computer Laboratory

University of Cambridge

A thesis submitted for the degree of

Doctor of Philosophy

Yet to be decided

Contents

List of Figures

Todo list

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

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

21, add GSMP protocol reference

31, Add some notes on SDNSIM

35, Add more details on the timestamp measurement label

45, add texttt for all OpenFlow messages

59, Maybe remove this section

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

95, describe some efforts to overcome middleboxes.

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

98, Add dropbox measurement study

98, report Dropbox problem

99, Add some reference to cloud controller

104, Mention that tactic is able to control OpenFlow also

104, Discuss weight of tactics

109, discnnected Signpost functionality

112, Mention OpenVPN ARP cache

Chapter 1

Introduction

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

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

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

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

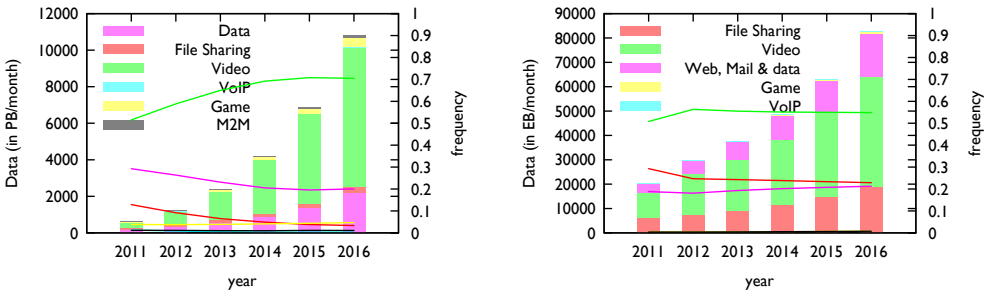
Application	rate	latency	jitter	# connections
web	0	0	0	0
video	0	0	0	0
p2p	0	0	0	0
voip	0	0	0	0
game	0	0	0	0

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

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

1.1 Motivation

Computer network evolution



(a) global mobile traffic trend

(b) global Internet traffic trend

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

One of the key mechanisms that formed the objective conditions for the digital revolution of our era, was the concept of computer networking. Initially, Computer networks aimed to provide a new communication architecture that would allow continuous communication over a redundant network, even when a significant number of links was destroyed. The main building block of computer networks is the idea of packet-switched networks Licklider [1963]. This idea gave birth to the pioneer of today's Internet, the *ARPANET* Mills and Braun [1987], allowing for the first time in

computing history communication between multiple computers over a mess network. The initial set of applications that were standardised where : e-mail Bhushan et al. [1973], ftp Bhushan [1972] and voice Cohen [1977]. This initial implementation was later replaced by the NSFNET in the 80's, which finally devolved in today's Internet. As part of this transition, the research community developed also the standards for the TCP/IP protocol suite Clark [1988], the default protocol to provide connectivity for the Internet.

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

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

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

High diversity is also observed on the properties of available mediums for computer networks. The properties of links is defined in the data link and physical layer of the OSI model. Currently, Ethernet is the predominant link layer protocol in the Internet.

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

Computer network ossification

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

Current network technologies were developed a number of years ago in order to develop standardized and generic mechanisms to interconnect research institutes. Although DARPA funded the idea of computer networks in order to develop new resilient communication mechanisms, the early adopters of the technologies were universities and research facilities. As a result, protocols were developed by computer scientist taking under consideration the properties of such environments. The TCP/IP protocol suite was develop during the transition of the ARPANET to NSFNET. Since then, the

TCP/IP protocol suite has been the default standard of the Internet. During the first period of the NSFNET, a number of competitive suites were developed which addressed in their specification the problem of extensibility *find references for OSI TP* protocol and ATM UNI* . Unfortunately, the increased design complexity made it difficult to develop high performance implementations, and they soon were declared obsolete by the network community. The TCP/IP protocol suite provided a fair split between simplicity and extensibility at that time.

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

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

Network Addressing : When the IP protocol was firstly deployed in the Internet, the size of the network was sufficiently small. Addressing was assigned based on a 32-bit integer space, split in byte aligned classes in order to permit aggregation at the forwarding entities. Within 10 years, the initial assumption over the size of classes was re-established through the classless Inter-domain routing (CIDR), in order to allow better utilisation of the IP space. Within 15 years though the initial assumption over the

size of the address space prooved also shortviewed, as IP addresses were not sufficient to cover the needs of hosts. A number of layer violations, like NATs, were widely used within the subsequent years in order to provide connectivity to the increasing number of end-hosts. In order to address this problem within the design of the network protocol, a revised version of IP has been proposed Deering and Hinden [1998] since 1998, but its deployment is slow, as the size of the current Internet makes it extremely difficult to replace IPv4 without significant connectivity problems and costs.

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

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

A number of problems that we experience with current network functionality can be traced back to the assumptions of the protocols. A number of clean slate approach

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

1.2 Contributions

1.3 Outline

1.4 Publications

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

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

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

This chapter presents existing approaches of network control in packet switched networks. We motivate the discussion over the problem of network control, describing the architecture of existing network devices and their discuss the physical limitations (Section 2.1). In addition, we present current control frameworks in modern production network and discuss their control limitations (Section 2.2). Further, we present three research network control schemes, namely Active Networking, Devolved Control of ATM Networks and Software Defined Networking (SDN), that redefine the network control abstraction and aim to address some of the limitations of existing approaches (Section 2.2). Finally, we discuss applications of these frameworks in specific network environments (Section 2.4).

2.1 Forwarding Devices

Packet networks have become a *'de-facto'* approach in communication networks. Their success relies, to a great degree, on the offered high medium utilization and low cost principle. As a result, this abstraction has been implemented over a large set of physical layer technologies and replaces tradition connection-oriented mechanisms (e.g. telephony, TV broadcast). In this section we focus on Ethernet and IP networks, the predominant protocols implementations for packet networks, and provide a hardware design overview of a *Switch*, a typical network forwarding device. Providing some high level insight on the architecture of a Switch, can highlight the existing lower bound on

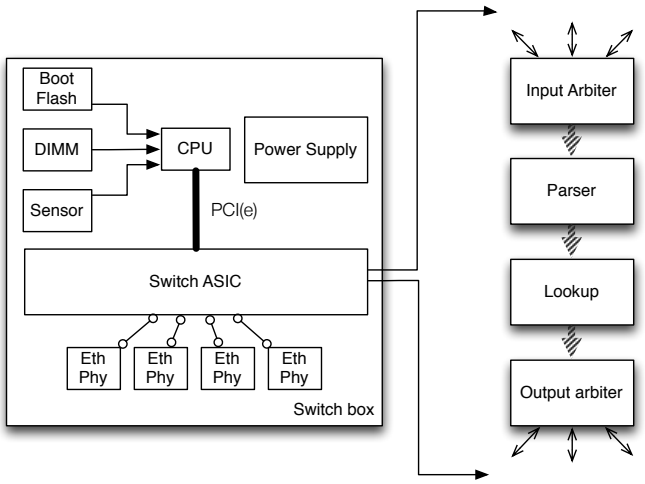


Figure 2.1: A generic architecture of a Switching device.

the functionality and extendibility of such devices.

Network switches are the main mechanism to multiplex Ethernet links. This device type provides collision-free connectivity between network segments. Switches have multiple functional roles in a network and as a result their architecture is highly diverse. For example, unmanaged switches provide a low cost non-configurable interconnection device for small networks, supporting usually a few 1 gig ports, while distribution switches are used in large enterprise networks to interconnect aggregation switches, and support multiple 10 or 40 Gig links. In this section we will use as an architectural example of a switch, the Top-of-Rack (TOR) switch. This switch type is used in datacenter networks to aggregation traffic between the edge and the distribution network segments. TOR switches can forward non-blocking high-rate traffic at line rate, using a single silicon chip. As presented in Figure 2.1, a typical switch device consists of the following components:

- *CPU*: Switch boxes contain a programmable CPU, which is used to run the control plane logic and provide reconfigurability to users. Switch vendors usually use low-power CPUs, such as SoC PowerPC chips or even ARM and MIPS CPUs. The CPU can access runtime memory module (Boot Flash and RAM), to store transient state, and persistent memory (e.g. SD Cards) through memory extension slots, to store log files and configuration. Such CPUs are sufficient to run simple switch control-plane functionality, but they are limited to support

intensive processing tasks.

- *Switch ASIC*: The Application-specific integrated circuit (ASIC) Broadcom [2013]; HP; Intel implements in hardware the forwarding plane of the switch. The capabilities of an ASIC are variable, depending on the vendor and the cost, but define the performance limits of the device to handle traffic. Implementation details of a ASIC silicon remain usually concealed, as they constitute an important asset for the producer. The development of a silicon design is a cost and time consuming process which takes years of development and testing and requires significant human and material resources. As a result, switch silicon innovation has a high latency to reach the production environment. Further, switch silicon face similar limitations as the CPU chips. The processing rate is upper bound by the transistor density and size.

  TOR-type switches can handle line-rate non-blocking traffic for a few ports, using a single ASIC silicon. An ASIC contains four important modules. The *input arbiter* module multiplexes and synchronizes packets from the Ethernet ports to the main processing pipeline of the silicon. The arbiter ensures non-preemptive packet processing, by bridging the rate mismatch between the silicon clock and the transmission rate of the link. In the main processing pipeline the ASIC requires a *protocol parsing* module and an *address lookup* module. The protocol parser extracts significant packet fields from a packet and the memory lookup module uses them to define the packet processing and forwarding operations. The lookup module relies on an interface with an external memory module in order to match protocol fields with forwarding policy. Memory modules have a trade-off between cost and access speed and there is a trade-off between memory cost and memory management complexity [1]. Finally, the *output arbiter* module is responsible to perform the required modifications to the packet and forward it to the appropriate output queue. Apart for the mentioned modules, a silicon can have additional processing modules to enable extended functionality such as access control list (ACL), flow statistics monitoring.

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

- *ASIC-CPU communication*: The switch CPU switching is interconnected with the ASIC over a PCI or PCI-express bus. The communication channel provides bi-directional communication: the firmware controls ASICs registers, to define forwarding functionality, and the ASIC propagates exceptions to the firmware. The channel provides sufficient bandwidth for control plane communication, but the rate is primarily defined by the speed of the CPU.

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

- *Ethernet ports* : The receipt and transmission of Ethernet packets is implemented as a separate hardware circuit. The module is responsible to implement the physical layer and MAC layer protocol and propagates packets to the ASICs using a Media Independent Interface (MII). The phy module usually contains small packet buffers to reduce packet losses for bursty traffic.

TOR switches are simple switch devices supporting traffic requirements on the edges of the network, and support lower to middle forwarding performance. As we move closer to the core of the network, the requirements for forwarding performance become higher and device complexity increases. Multi-chassis switches and routers cannot contain the required functionality in a single silicon. Packet processing is distributed between multiple silicons and complex clos interconnection crossbars are used to provide non-blocking terabit backplane capacity Juniper [2012]. In addition, providing fast address lookup requires multiple distributed buffers and cache coherent protocols to ensure state consistency Cisco. In this classes of devices, the control plane abstraction has a complex and slow translation to appropriate management of the silicon and flexibility is restricted.

2.2 Forwarding Control in Production Networks

Dynamic network control has been an important functionality for computer network technologies. Forwarding devices can reconfigure their forwarding logic, when the

network state changes, and thus provide end-to-end connectivity, adjusted to the network environment. Existing standardized approaches aim for *distributed*, *autonomous* and *resilient* network control and are influenced extensively by the end-to-end principle of the Internet. A switch or a router control functionality requires only local forwarding configuration. The device at run-time uses a number of well-defined distributed protocols that allow the reconstruction of the global network configuration, by configuration sharing with other devices. Using the global network configuration, the forwarding logic can calculate an optimal forwarding policy and maximize network performance. Because the state is constructed collectively, the network provides log-term resilience. If a link is disconnected, the distributed protocol can propagate this information across the network and devices can recalculate the optimal policy, while if a device is rebooted, it can rebuild its global view using only its local configuration. In this section we present existing production-level protocols that enable distributed network control. We focus on the Data-Link and Network layer protocols as these layer define the hop-by-hop forwarding.

2.2.1 Data-Link Layer Control

Data-Link layer control protocols provide loop-detection, and VLAN and QoS automation. Loop detection is used to avoid packet loops in networks with redundant connectivity, as Ethernet doesn't provide timeout functionality. Spanning Tree protocol(STP) was a first attempt to addressed this problem, standardised by IEEE in 802.1D-1998 IEEE [1998]. The protocol implemented the distributed spanning tree construction algorithm presented in Perlman [1985]. The algorithm uses broadcast messages to discover a spanning tree over the graph of the network, with respect to a common root switch and controls a state machine per port, which disables packet forwarding when a link is not part of the spanning tree. Because the initial definition of the protocol was slow in tree re-calculation after a network change, IEEE introduce the Rapid Spanning Tree Protocol (RSTP), an evolve version of STP, in IEEE [2004]. In addition, a modified version of the protocol was defined to address the requirements of VLAN functionality. VLAN tagging can creates multiple subgraphs over the network. The protocol runs individual STP algorithms each subgraph to construct individual spanning trees and reduce unnecessary port blocking. This functionality is provided

by IEEE Multiple Spanning Tree Protocol (MSTP) IEEE [2005], while Cisco has developed a series of proprietary protocols Cisco [2012b,c]. Finally, IEEE has recently defined an advanced STP protocol that enables switches to discover and use in parallel redundant network links, in the IEEE 802.1aq standard IEEE [2006].

In terms of automatic configuration, network vendors and standardization bodies have developed a number of protocols that can disseminate information to neighbouring nodes and ease device management. Such protocols have high churn between vendors and each vendor defines its custom standard. In this class of protocols we consider Link Layer Discovery Protocol (LLDP), supported by IEEE, Cisco Discovery Protocol (CDP), Extreme Discovery Protocol (EDP) and Nortel Discovery Protocol (NDP). Finally,in order to reduce the administration burden of VLAN configuration for inter-switch trunk links, Cisco has developed the VLAN Trunking Protocol (VTP).

In the class of Data Link layer protocols we also consider the MPLS protocol Rosen et al. [2001]. The MPLS technology tries to bridge the gap between virtual-circuit technologies, like ATM, and packet-switched networks. MPLS reduces the ATM control plane complexity and removes the limiting small cell size. MPLS forwarding uses tunnel labels, that simplify the lookup process and reduces memory requirements. MPLS is used extensively in the distribution layer of large networks. In order to automate the configuration of MPLS circuits, the protocol use the LDP protocol Andrerson et al. [2007] to setup automatically labels across the network. In addtion, IETF defined a mechanism to translate RSVP resource requests to MPLS tunnel configuration Awduche et al. [2001]. Because MPLS doesn't provide resource control, the IETF defined the *Autobandwidth* mechanism. Autobandwidth monitors tunnel bandwidth requirements and recalculates paths when bandwidth requirements change. A study on a deployment of this technology in the MSN network though has highlighted that the allocation policy is under-optimal for significant periods of the network functionality Pathak et al. [2011].

2.2.2 Routing control protocols

In the network layer, modern network technologies exercise control using routing protocols. Routing protocols follow a similar approach to Data Link layer protocols and provide mechanisms to disseminate local configuration to the network. Routing proto-

cols are classified in three categories: *Link State*, *Distance Vector* and *Path Vector*. Link state protocols theory build on-top of the Djikstra algorithm Dijkstra [1959]. Routers disseminate their local forwarding configuration to the rest of the network. Using this global state exchange, each router is able to construct the global connection graph. Each router can use the graph to calculate the minimum spanning tree of the network, using Djikstra's algorithm. Currently there is a plethora of Link State protocol specification in the network community. IETF has developed the OSPF protocol Moy [1998], an Ipv4 specific routing protocol, while the Open Systems Interconnection (OSI) organisation has defined the IS-IS protocol Oran [1990], a network layer agnostic routing protocol. Link state routing protocols provide provide high flexibility to define the optimisation function of the routing system. Each router has view over the complete graph of the network and router can propagate multiple link performance indexes (e.g. link load, link speed). Nonetheless, the optimization function must be homogeneous across the network, in order to avoid routing loops.

Distance Vector routing protocols employ a different approach to provide efficient global routing policies. Routing table is constructed using information from adjacent routers. Network changes are slowly propagated across the network, through point-to-point information exchanges, until all routers converge. Distance Vector protocols use the Belman-Ford algorithm Bellman [1956] to achieve routing optimality. IETF developed the RIP protocol Malkin [1998] to implement Distance Vector routing, while Cisco has developed the proprietary IGMP protocol Rutgers [1991]. Distance Vector protocols are less extensible than Link State routing protocols, but can be implemented with smaller computational and memory requirements.

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

Due to their distributed nature, routing protocols provide long-term routing resilience, while their mathematical foundation is to provably correct. Nonetheless, the control abstraction of these protocol is not a good fit for administrator to exercise dy-

namic control in the network. For example, it is intractable for a network manager to speculate on the impact of network policy changes. In 2008 the global Internet was severely affected by a BGP misconfiguration from the Pakistani national ISP in an effort to control traffic from YouTube service Brown [2008]. In addition, routing inconsistencies during routing changes can impact significantly network performance. For OSPF, authors in Watson et al. [2003] monitor OSPF functionality in a regional ISP and report high churn in routing even during periods without significant network events, while the latency to converge after a network change is on average on the order of seconds. Similar results have been observer in BGP. In Kushman et al. [2007] the authors highlight a strong correlation of BGP instability and VoIP performance.

2.3 Programmable Network Control

Because of the limitation of current standardized network control frameworks, the research community over the years has proposed a number of interesting approaches for network control evolution. In this section we present the three most significant and complete approach to this problem, namely Active Networks, Devolved Control of ATM Networks and Software Defined Networking.

2.3.1 Active Networks

The problem of protocol ossification in the network layer and the requirement for network evolution, was highlighted by the network research community since mid 90's. In O'Malley and Peterson [1992], the authors argue on the generality of the 7-layer OSI network abstraction. They claimed that the networks functionality should be more dynamic and forwarding devices should be able to accommodate variable number of protocol layers. In their work they present an extensible protocol modeling framework to layer protocol processing. Motivated by these observations, DARPA funded the *Active Networks* project DARPA, in an effort to develop next generation network devices that can accommodate seamlessly new protocol functionality.

Active networks modify the default packet processing mechanism and introduce the notion of *capsules*. Capsules are network packets that carry data, as well as, processing code. On each hop, the default packet processing logic is extended with the

code contained in the capsule, thus redefining network functionality. This network processing approach provided user-driven protocols upgrades, without a requirement for devices upgrade. In addition, the development of novel programming languages, like Java, provided the building blocks to implement code distribution frameworks.

Research in active networks tried to define two mechanisms : the *capsule API and format* and the *switch architecture*. In terms of capsule format, the active network community defined the Active Network Encapsulation Protocol (ANEP) Alexander et al. [1997a]. ANEP, through its format specification, made a clear separation between functionality and data. An Active Node using a new protocol would inject firstly new processing logic, using code capsules, and then send the data stream, as data capsules. The format was adopted by the ANTS and Sprocket frameworks. Sprocket Schwartz et al. [2000] was an effort to develop an efficient capsule programming language by BBN technologies. The language uses a restricted set of the C language, avoiding any security vulnerable structures like pointers, while it introduced native support for SNMP browsing. Sprocket compiled source code to compact MIPS assembly. Each switch would receive code capsules and run the code in a MIPS virtual machine. The VM could not persist any state on the switch, but it was able to modify the data section of a packet. Sprocket provided secured injected code with a public key secure hash of the code binary, but without sufficient details on the authentication logic. ANTS Wetherall et al. [1998] was an attempt by the Active Networks group in MIT to develop a sufficient capsule programming environment. ANTS uses a restricted version of the Java language to program capsules, because of the inherent object serialization functionality of Java, while the processing logic is defined through a small set of Java interfaces. ANTS permits injected code to persist flow state on a switch and modify packet content. An interesting extension of the ANTS framework was the code dissemination mechanism. An end-node deployed new protocol functionality solely to the local Internet gateway, and the code is forwarded along with the packets to each next-hop. Finally, The university of Pensylvania active networks group developed its own approach to capsule programming. PLAN Hicks et al. [1998] used a restricted version of the OCaml language to abstract capsule access to the local switch state and implement custom forwarding logic. PLAN did not follow the ANEP packet format. A PLAN capsule contains code and data and the code section replaces the network header information. In order to secure switch infrastructure, the language disallowed

packet data modification or switch state persistence. Ultimately, the language provided a framework for programmable control plane.

In terms of Active Network platform architecture, the community developed a number of architectures, addressing multiple aspects of efficiency in capsule processing. University of Pensylvania proposed the SwitchWare Execution Environment (EE) Alexander et al. [1998a]. The architecture defined an OCaml module framework for capsule processing. A core focus of the SwitchWare EE was strong multi-dimension security for active networks, both on the local system, as well as, during capsule execution. SwitchWare used SANE Alexander et al. [1998b], a trustworthy operating system, to secure the functionality of the switch, and build on top of its security primitive higher level of trust. The platform addressed issues regarding secure execution and authentication, as well as, capsule code verification. PLANet Hicks et al. [1999] and Active Bridge Alexander et al. [1997b] used the SwitchWare framework to implement novel functionality in active networks.

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

The CANEs project Chae and Zegura [2002] from Georgia Tech Active Networks groups followed a different approach for the problem and designed a system which enabled multi-protocol packet processing. CANEs defined a number of abstractions, which enabled active network applications to stack multiple protocol processors on the forwarding path. CANEs project build on top of the Bowman Switch OS Merugu et al. [1999] and established a simple and efficient abstraction over the Switch resources.

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

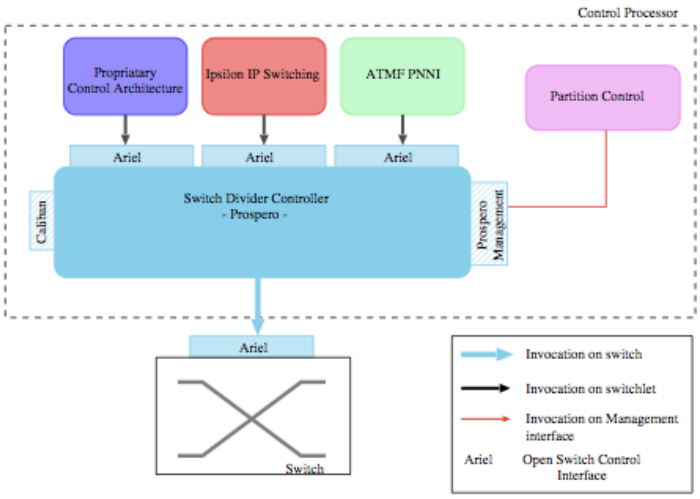


Figure 2.2: Tempest switch architecture van der Merwe [1998]

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

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

2.3.2 Devolved Control of ATM Networks

add GSMP protocol reference An effort for network programmability in ATM network was also developed in the University of Cambridge, as part of the DCAN (Devolved Control ATM Networks) project project [2000]. The system, called Tempest Rooney et al. [1998], aimed to provide dynamic network control and resource virtualization. Tempest provides a clear separation between the control and forwarding plane of a network device. The control plane of network devices, is centralised in a single programmable entity, which implements intelligent and dynamic forwarding. The implementation of Tempest is logically divided between three abstractions, depicted in Figure 2.2. The Figure presents how a single switch is able to function in parallel as an IP router, an ATM switch and a Hollowman controlller Rooney [1997], a devolved ATM control framework.

Prospero Switch Divider The *Prospero* abstraction provides a mechanism to virtualise ATM switch resources. Prospero provides a resource control interface, which can divide switch resource between multiple virtual switches, called *switchlets*. A switchlet controls a subset of the switch ports, VCI and VPI mappings, packet buffer and bandwidth. For packet buffer and bandwidth virtualization, Prospero uses the ATM defined QoS principles. In addition, Prospero is responsible to map the control interface, exposed to controllers, with the underlying switch functionality.

Ariel Switch Independent Control Interface Switchlets expose forwarding control through the Ariel Interface. Ariel Interface organises network control through five control objects: *Configuration*, *Port*, *Context*, *Connections Statistics* and *Alarms*. Configuration provides details for the switch configuration, Port provides primitive controllability of ports (e.g. state, loopback functionality), Context enables QoS policy control, Connections expose control of VPI/VCI mappings, Statistics exposes packet and byte counters and Alarms push state change notifications to the controller. Any Tempest switch runs an Ariel server and translates Ariel request to Prospero control requests.

Caliban Switch Management Interface Apart from control capability support, Tempest also supported evolved network management, through the Caliban interface. The

interface functionality is similar to the SNMP protocol. Caliban provides fine level SNMP-style information, but can also provide higher level aggregation operations over the switch state.

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

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

2.3.3 SDN

A recent attempt to develop a framework for distributed and scalable control is the Software Defined Networking (SDN) paradigm. SDN Foundations [2012] propose the clean separation between the control and the forwarding plane of a network device. The control plane logic is implemented in a separate programmable device and a well defined protocol establishes the inter-connection between the two devices.

The SDN approach was highly motivated by relevant earlier attempts in network programmability(Active Networks and DCAN), but follows an evolutionary approach. SDN supports existing network data protocols but enables the development of custom forwarding mechanisms. The paradigm is motivated by two observations in current network technologies. Firstly, the evolution of computer networks has reduced the applicability of the multi-layer network abstraction. Production networks contain middleboxes (e.g. firewall, NAT, layer-5 switches) that operate on multiple layers of the network and engage extensively in layer violation. Such devices cannot integrate with

existing control mechanisms, while they remain transparent by the end-hosts. The SDN paradigm aims to provide a new unifying cross-layer control model that can accommodate such control approaches. Secondly, although network complexity has increased, network devices still use stand-alone configuration mechanisms (e.g. remote logic, CLI interfaces) and network policies must be decomposed by network administrators in individual network device policies to fit the existing control abstraction. In order to modify network-wide configuration, network administrators must login over telnet or use a web interface to configure each device manually. In addition, network vendors develop proprietary control protocols which reduce device interoperable. SDN enables management centralisation, while the flexibility of the control abstraction can create new innovating control mechanisms that match network environment requirements.

The SDN paradigm was highly successful in the network community and within a few years since its introduction, network vendors engaged in the development of production-level implementations. In addition, the research community has developed a number of interesting SDN architectures which address several network problems. The OpenFlow protocol is currently the pre-dominant implementation of the SDN paradigm.

2.3.3.1 OpenFlow protocol

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

A representation of a simple OpenFlow topology is presented in Figure 2.3. A minimum OpenFlow topology consists of two entities: the *Controller* and the *Switch*. The Controller implements and disseminates the control logic of the network to the switches over a control channel. A controller can control multiple switches. In addition, the network community has developed a number of programming environments

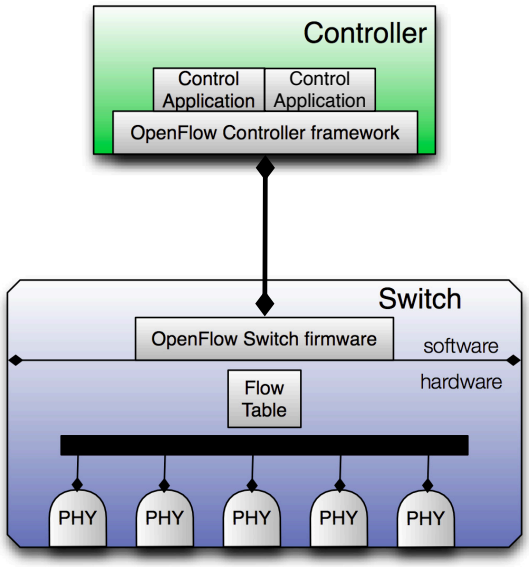


Figure 2.3: A schematic representation of a basic OpenFlow setup

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

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

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

Table 2.1: OpenFlow tuple fields

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

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

Table 2.2: OpenFlow available packet actions

to develop OpenFlow control applications. Such programming environments provide a higher level abstraction over the OpenFlow protocol and reduce the exposure of the developer to the run-time details Berger [2012]; Gude et al. [2008b]; Switch [2013].

The switch entity is responsible to transform OpenFlow protocol operations in data path policy modifications. The OpenFlow functionality is split between the data and control layer of the switch. In the data layer, the protocol implements a simple packet processing algorithm. For each packet, the hardware extracts a number of fields and matches them against the entries of a *Flow Table*. The Flow Table is a memory abstraction which contains forwarding policy. It comprises of flow entries which conceptually consist of three parts: the flow tuple, the flow statistics and the flow action list. The flow tuple synthesizes a flow description using all the significant packet header fields. We present the list of fields in Table 2.1. The protocol associates with each flow tuple a flow mask, which permits field wildcarding in a flow definition. Flow statistics store flow packet and byte counters. The action list, contains the list of actions that should be applied on every matched packet. We present in Table 2.2 the packet processing actions defined by the OpenFlow protocol. On a packet reception, if a matching flow is found in the flow table, then the flow statistics are updated and the action list is applied on the packet. If there aren't any matching flow entries, the packet propagates as an exception to the control layer of the switch. The control plane of the switch implements the translation of the OpenFlow protocol into data path modifications. It provides access and modification functionality to the switch configuration and the flow table. The switch is able also to propagate autonomously information and exceptions to the controller.

The ensemble of a flow table and a set of switch ports comprises an OpenFlow *datapath* and is the core abstraction for network control. The protocol defines a number of protocol message types to manage datapath resources. OpenFlow control operations can be grouped in two categories: *Forwarding control* and *Switch configuration*. In the rest of the section we present how this functionality is addressed by OpenFlow messages and how this mechanism is evolved over the different versions of the protocol. We focus our protocol presentation over the 1.0 version of the protocol. ONF has releases three revisions of the protocol, version 1.1, 1.2 and 1.3, which change significantly the protocol specification. Nonetheless, the majority of production systems are predominantly version 1.0 compliant.

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

Switch configuration Apart from the control of the flow table, the OpenFlow protocol provides switch configuration capabilities. A controller can use the `switch_config_req/switc` message types, to discover switch OpenFlow operation support. In addition, the protocol provides capabilities to control port state with the `port_mod` message type. The switch is also able to notify the controller when a port changes its state (e.g. link is detected to be inactive in the physical layer) with the `port_status` message. In the recent revisions of the protocol, the steering committee has introduced the capability to control per port traffic shaping queues.

Protocol evolution Since the specification of the version 1.0 of the protocol, the steering committee of the protocol has produced 3 non-backward compatible revisions of the protocol. The updates in the protocol are motivated both by the osmosis between the hardware and software OpenFlow community, as well as, by the introduction of new deployment use cases.

Version 1.1 of the protocol introduces a number of new features for integration of the protocol with the silicon functionality. Firstly, the default packet processing algorithm is modified. A switch can accomdate multiple flow tables and the lookup process searches sequentially each of the tables for flow matches. The packet must have a match in each of the tables otherwise a `pkt_in` message is generated. In addition, in order to persist partial results between tables, the protocol define a 64-bit metadata field and the action list is augmented with operations over the metadata field, as well as, over the table search process (e.g. terminate lookup, skip table). Secondly, the protocol introduces a primitive to express multipath support in the protocol. This functionality is exposed through a group table abstraction. Group table entries contain a set of buckets containing flow actions and an assigned output port. A controller in the action of a flow can forward packet to a group entry. The goup selection action can select to forward the packet either to all the buckets of the entry (ALL) or select randomly a bucket, in a manner similar to the Equal Cost Multiple Path routing (ECMP) Hopps [2000] functionality, (SELECT) or select a specific bucket (INDIRECT) or forward packet to the first group entry for which the output port can forward packet (FAST_FAILOVER). Thirdly, the protocol introduces MPLS matching support and manipulation.

Version 1.2 of the protocol augments further protocol functionality. This revision introduces support for IPv6 and Provider Bridge Network (PBB - Mac-in-Mac) traffic. In addition the protocol defines a new TLV format for flow Match definition, in order to relax the OpenFlow tuple definition. Finally, the protocol defines a model to control multiple controller access to the switch. Multi-Controller schemes provides resilience to the control of the network.

In Version 1.3 the protocol definition introduces protocol support of QoS control. The protocol defines a new table abstraction, the metering table, which contains queue definitions. In addition, this protocol definition introduces the idea of multiple parallel control channels towards the same controller, in order to paralelise `pkt_in` transmission, and defines a mechanism to suport message fragmentation.

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

2.4 Control Plane Applications

2.5 Conclusions

Chapter 3

SDN control mechanism evaluation

In our effort to develop scalable network control, we employ extensively the SDN design approach and the OpenFlow protocol abstraction to develop our applications. The SDN protocol provides two core functional properties that make it an excellent candidate. On one hand, the clean separation of the control and the forwarding plane in a network is inherently backwards compatibility with existing network applications and interoperable with existing network devices. The evolution of the control plane of a network doesn't affect the support of forwarding plane protocols, while SDN can be deployed gradually in existing networks without forwarding performance penalties. As a result, a number of network vendors provide production-level support of OpenFlow control interfaces. Secondly, the OpenFlow control abstraction is sufficiently generic to support a diverse set of control approaches. The protocol provides support both reactive and proactive control schemes, while the flow definition granularity is dynamically controlled to match the environment requirement. In the following chapter of this thesis we employ the OpenFlow protocol to address two diverse network scalability problems.

In this chapter we present an extensive performance analysis of the control scalability of available SDN technologies. Our exploration aims to provide an in depth presentation of the limitations of existing implementation efforts and understand their impact in forwarding plane performance, as well as, provide a set of tools that enables SDN developers to study the performance of their network designs. The work focuses on implementations of version 1.0 of the OpenFlow protocol, the only production-level protocol instantiation of the SDN paradigm. We conduct our analysis using

two measurement platforms: OFLOPS and SDNSIM. OFLOPS is a high precision OpenFlow switch micro-benchmark platform. Using OFLOPS, we implement a set of benchmarking to characterise the performance of elementary OpenFlow protocol interactions. SDNSIM is a macro-benchmark OpenFlow platform, which extends the Unikernel framework Madhavapeddy et al. [2013] to support large scale OpenFlow-based network simulations and emulations. Using SDNSIM, developers are able to import OFLOPS switch profiles in their experiment and test the performance of their SDN design.

In this Chapter, we present the motivations (Section 3.2) and the design overview of the OFLOPS platform (Section 3.3). We select a number of off-the-self OpenFlow switches (Section 3.4) and run against them a number of measurement experiments, in order to assess the elementary protocol interaction performance (Section 3.5). Furthermore, we present the SDNSIM platform (Section 3.6) and its design approach (Section 3.8). Finally, we assess the performance of the SDNSIM implementation along with a measurement study of control scalability over the fat-tree topology (Section 3.9) and conclude (Section 3.1).

3.1 OFLOPS Conclusions

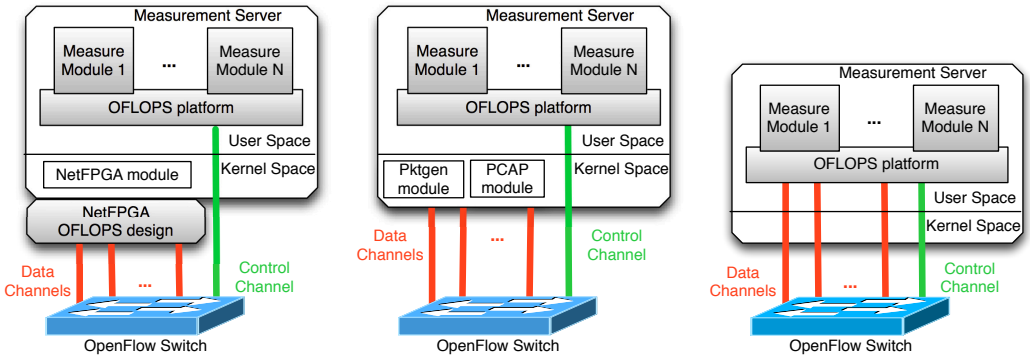
Add some notes on SDNSIM We presented, OFLOPS, a tool that tests the capabilities and performance of OpenFlow-enabled software and hardware switches. OFLOPS combines advanced hardware instrumentation, for accuracy and performance, and provides an extensible software framework. We use OFLOPS to evaluate five different OpenFlow switch implementations, in terms of OpenFlow protocol support as well as performance.

We identify considerable variation among the tested OpenFlow implementations. We take advantage of the ability of OFLOPS for data plane measurements to quantify accurately how fast switches process and apply OpenFlow commands. For example, we found that the `barrier` reply message is not correctly implemented, making it difficult to predict when flow operations will be seen by the data plane. Finally, we found that the monitoring capabilities of existing hardware switches have limitations in their ability to sustain high rates of requests. Further, at high rates, monitoring operations impact other OpenFlow commands.

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

3.2 Network Control Micro-Benchmark and Characterisation

Despite the recent introduction of the SDN paradigm, the research community has already proposed a wide range of novel control architectures Handigol et al. [2009]; Sherwood et al. [2010]; Yu et al. [2010]. These architectures address significant problem of modern networking, but their deployment in production environments is not straightforward. Computer network have become a vital asset for modern enterprises, and high availability and performance are critical. Modifying established network control mechanisms must ensure these requirements and requires extensive testing and performance characterisation. An example of an early OpenFlow-based SDN deployment that faced significant problems in a production environment is reported in Weissmann and Seetharaman [2011]. The authors describe their experience in deploying the first OpenFlow production network in the Computer Science department in Stanford University. In their article, they point out that the initial deployment exhibited significant performance and reliability problems. The deployed hardware switching platform couldn't support OpenFlow message processing at high rates. For example, in the morning a significant number of users would plug simultaneously their computers to the network, creating a large burst of new network flows. Because the reactive control approach they employed requires the generation of a `pkt_in` message for each new connection, the switch control CPU processing capacity couldn't cope with the high utilisation and either a number of flows would be dropped or the control channel would become unresponsive. In the SDN application development toolchain, there is a lack of an established and global mechanism to assess switch performance. For traditional switch devices, the network community employs as performance metrics, the switch fabric non-blocking processing capacity and latency and MAC table size. In contrast, the control decoupling of the SDN paradigm expands the ways that the controller interacts with a switch device, while these interactions become a significant requirement for the functionality of the switch. As a result, the performance characterisation of



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

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

SDN forwarding devices becomes non-trivial.

In order to address this issue we developed OFLOPS [1], a measurement framework that enables rapid development of performance tests for both hardware and software OpenFlow switch implementations. To better understand the behaviour of the tested OpenFlow implementations, OFLOPS combines measurements from the OpenFlow control channel with data-plane measurements. To ensure sub-millisecond-level accuracy of the measurements, we bundle the OFLOPS software with specialized hardware in the form of the NetFPGA platform[2]. Note that if the tests do not require millisecond-level accuracy, commodity hardware can be used instead of the NetFPGA Arlos and Fiedler [2007].

3.3 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 dis-

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

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

close implementation details. In order to understand the performance impact of an experiment, multiple input measurement channels must be monitored concurrently, such as, data and control channels. Further, current controller frameworks, like Gude et al. [2008a]; Switch [2013], target production networks and incur significant measurement noise. Such controllers employ asynchronous processing libraries and multi-thread execution, to maximize the aggregate control channel processing throughput, and provide high-level programing interfaces, that require multiple data structure allocation and modification during packet processing. The processing time of a specific OpenFlow message, especially at high rates, is subject to multiple aspects, like the thread scheduling policy and the asynchronous library execution logic Jarschel et al. [2012]. Measurement noise suppression in the control plane requires from the controller framework to minimize message processing and delegate control to the measurement module. Finally, sub-millisecond precision in software is subject to unobserved parameters, like OS scheduling and clock drift. The result of these challenges is that meaningful, controlled, repeatable performance tests are non-trivial in an OpenFlow environment.

The OFLOPS design philosophy aims to develop a low overhead abstraction layer that allows interaction with an OpenFlow-enabled device over multiple data channels. The platform provides a unified system that allows developers to control and receive information from multiple control sources: data and control channels as well as SNMP to provide specific switch-state information. For the development of measurement experiments over OFLOPS, the platform provides an event-driven, API that allows 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 3.3. Details of the OFLOPS programming model can be found in the API manual OFLOPS [2011].

The platform is implemented as a multi-threaded application, to take advantage of modern multicore environments. To reduce latency, our design avoids concurrent access controls: we leave any concurrency-control complexity to individual module implementations. OFLOPS consists of the following five threads, each one serving specific type of events:

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

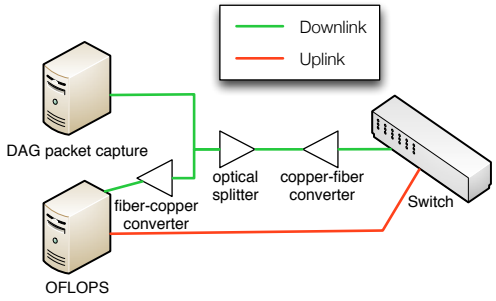


Figure 3.2: Measurement topology to evaluate capturing mechanism precision.

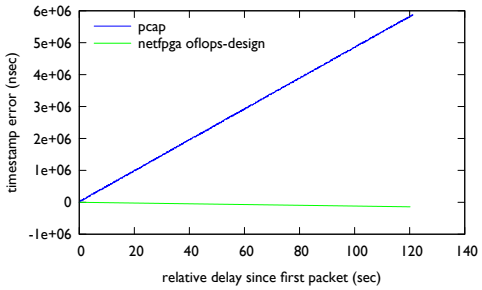


Figure 3.3: Evaluating timestamping precision using a DAG card.

2. Data Packet Capture: data plane traffic interception.

3. Control Channel: controller events dispatcher.

4. SNMP Channel: SNMP event dispatcher.

5. Time Manager: time events dispatcher.

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

Add more details on the timestamp measurement label In order to assess the accuracy of the traffic capturing mechanism, we use the topology depicted in Figure 3.2 to measure precision loss in comparison to a DAG card Endace. To calculate precision, we use a constant rate 100 Mbps probe of small packets for a two minute period. The probe is duplicated, using an optical wiretap with negligible delay, and sent simultaneously to OFLOPS and to a DAG card. In Figure 3.3, we plot the differences of

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

3.4 Measurement setup

At the end of 2009, the OpenFlow protocol specification was released in its first stable version 1.0 OpenFlow Consortium [2009b], the first recommended version implemented by vendors for production systems. The switch performance analysis was contacted in collaboration with T-labs in Spring of 2011. During that period, the introduction of OpenFlow support in commodity switches was limited and only a small number of devices provided production-level support for the protocol. Using OFLOPS, we evaluated OpenFlow-enabled switches from three different switch vendors. Vendor 1 has production-ready OpenFlow support, whereas vendors 2 and 3 at this point only provide experimental OpenFlow support. The set of selected switches provided a representative but not exhaustive sample of available OpenFlow-enabled top-of-rack-style switching hardware. Details regarding the CPU and the size of the flow table of the switches are provided in Table 3.1. In addition the switching fabric in the vendor hardware specification reports similar non-blocking capacity and packet processing rates.

OpenFlow is not limited to hardware. The OpenFlow protocol reference is the software switch, Open vSwitch Pettit et al. [2010], an important implementation for production environments. Firstly, Open vSwitch provides a replacement for the poor-performing Linux bridge Bianco et al. [2010], a crucial functionality for virtualised operating systems. Currently, the Xen platform uses Open vSwitch to forward traffic between VMs in the Dum0, a configuration which is inherited by major Cloud providers, like Amazon and Rackspace. Secondly, several hardware switch vendors use Open vSwitch as the basis for the development of their own OpenFlow-enabled firmware. Open vSwitch development team has standardised a clean abstraction 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. We study Open vSwitch

alongside our performance and scalability study of hardware switches. Finally, in our comparison we include the OpenFlow switch design for the NetFPGA platform Naous et al. [2008]. This implementation is based on the original OpenFlow reference implementation OpenFlow Consortium [2009a], extending it with a hardware forwarding design.

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

Table 3.1: OpenFlow switch details.

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

3.5 Switch Evaluation

As for most networking standards, there are different ways to implement a given protocol based on a paper specification. OpenFlow is no different in this regard. The current OpenFlow reference implementation is Open vSwitch Pettit et al. [2010]. However, different software and hardware implementations may not implement all features defined in the Open vSwitch reference, or they may behave in an unexpected way. In order to understand the behaviour of switch OpenFlow implementation, we develop a suite of measurement experiments to benchmark the functionality of the elementary protocol interactions. These tests target (1) the OpenFlow packet processing actions (Section 3.5.1), (2) the packet interception and packet injection functionality of the protocol (section 3.5.2), (3) the update rate of the OpenFlow flow table along with its impact on the data plane, (Section 3.5.3) (4) the monitoring capabilities provided by

OpenFlow (Section 3.5.4), and (5) the impact of interactions between different OpenFlow operations (Section 3.5.5).

3.5.1 Packet modifications

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

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

We observe significant differences in the performance of the hardware switches due in part to the way their firmware implements packet modifications. Switch1, with its production-grade implementation, handles all modifications in hardware; this explains its low packet processing delay between 3 and 4 microseconds. On the other hand, Switch2 and Switch3 each run experimental firmware that provided, at the time, only partial hardware support for OpenFlow actions. Switch2 uses the switch CPU to perform some of the available field modifications, resulting in two orders of magnitude higher packet processing delay and variance. Switch3 follows a different approach; all packets of flows with actions not supported in hardware are silently discarded. The

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

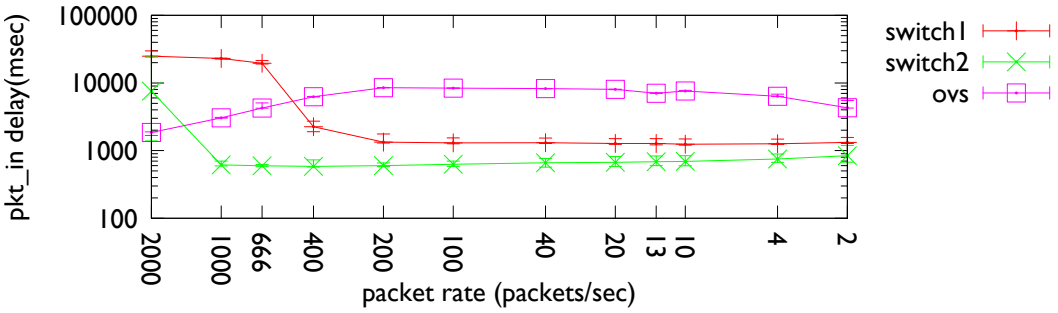
Mod. type	Switch 3			NetFPGA		
	med	sd	loss%	med	sd	loss%
Forward	5	0	0	3	0	0
MAC addr.	-	-	100	3	0	0
IP addr.	-	-	100	3	0	0
IP ToS	-	-	100	3	0	0
L4 port	-	-	100	3	0	0
VLAN pcp	5	0	0	3	0	0
VLAN id	5	0	0	3	0	0
VLAN rem.	5	0	0	3	0	0

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).

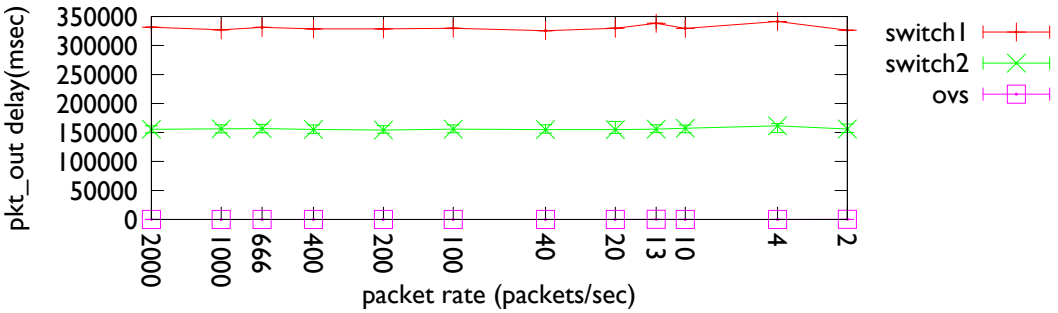
performance of the Open vSwitch software implementation lies between Switch1 and the other hardware switches. Open vSwitch fully implements all OpenFlow actions. However, hardware switches outperform Open vSwitch when the flow actions are supported in hardware.

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

3.5.2 Traffic interception and injection



(a) `packet_in` message latency



(b) `packet_out` message latency

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

OpenFlow protocol permits a controller to intercept or inject traffic from the control plane. Packet interception enables reactive control applications, while packet injection permits control application interaction with network hosts. Nonetheless, the interception mechanism in OpenFlow has been characterised as a significant bottleneck for the control plane in early OpenFlow protocol deployments Kobayashi et al.. This is a direct consequence of the silicon design in current OpenFlow switches, that develop such functionality over a low-bandwidth exception-notification channel. In order to characterise these functionalities, we design two simple experiments. For packet interception, we remove all entries from the switch flow table and send a measurement probe of small packets (100 bytes) on one of the data channels. We measure the delay between the time the packet was transmitted on the data channel and the time the controller received the equivalent `packet_in` message. For packet injection, we transmit

packet_out messages over the control channel and measure the delay to receive the packet on a data channel. In Figure 3.4, we plot the median packet processing latency for packet_in and packet_out messages. We omit in this experiment Switch 3 as these functionalities incur high CPU utilisation and, after a few seconds of traffic, the control channel becomes unresponsive. For packet_out messages, all implementations rate limit their transmission through the TCP advertised window of the control connection and as a result the latency is near constant. We observe that hardware switch exhibit hight latency (150 msec for Switch2 and 350 msec for Switch1), in comparison to Open vSwitch (approximately 0.1 msec). For packet_in messages, we observe adiverse behaviour between hardware switches at high packet rates. For Switch1, packet loss and latency becomes note-worthy beyond 400 packets/sec, while the switch can process up to 500 packets/sec. For Switch2 latency and packet loss are significantly lower and stable. Switch2 incurs high processing latency beyond 2000 packets/sec. Open vSwitch, has a high but stable latency for any tested data rates.

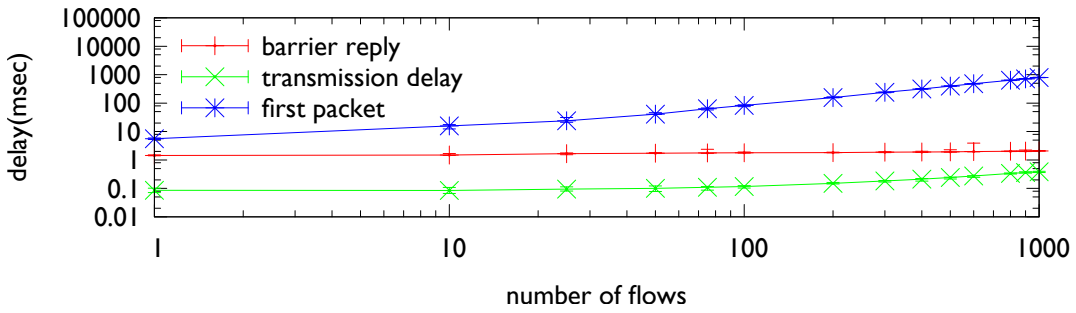
3.5.3 Flow table update rate

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

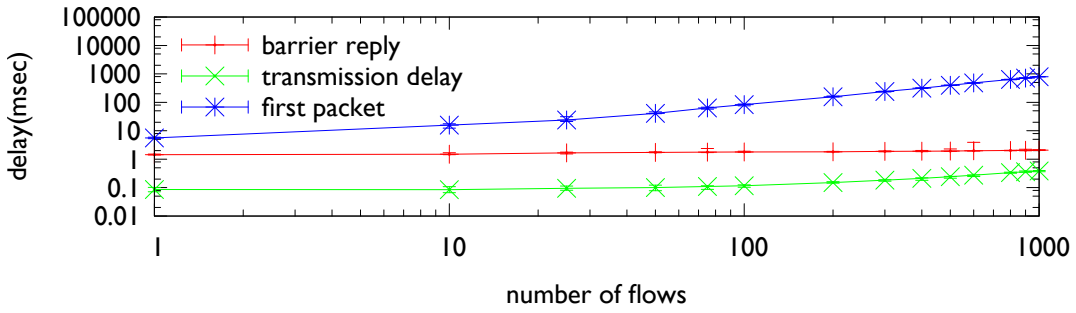
OpenFlow provides a mechanism to define barriers between sets of commands: the barrier command. According to the OpenFlow specification OpenFlow Consortium [2009b], the barrier command is a way to be notified that a set of OpenFlow operations has been completed. Further, the switch has to complete the set of operations issued prior to the barrier before executing any further operation. If the OpenFlow implementations comply with the specification, we expect to receive a barrier notification for a flow modification once the flow table of the switch has been updated, implying that the change can be seen from the data plane.

We check the behavior of the tested OpenFlow implementations, finding variation

among them. For Open vSwitch and Switch1, Figure 3.5 shows the time to install a set of entries in the flow table. Switch2 and Switch3 are not reported as this OpenFlow message is not supported by the firmware. For this experiment, OFLOPS relies on a stream of packets of 100 bytes at a constant rate of 100 Kpackets/sec (10Mbps) that targets the newly installed flows in a round-robin manner. The probe achieves sufficiently low inter-packet periods in order to accurately measure the flow insertion time.



(a) Open vSwitch (log-log scale)



(b) Switch1 (log-log scale)

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

In Figure 3.5, we show three different times. The first, *barrier notification*, is derived by measuring the time between when the **first insertion command** is sent by the OFLOPS controller and the time the `barrier` notification is received by the PC. The second, *transmission delay*, is the time between the first and last flow insertion commands are sent out from the PC running OFLOPS. The third, *first packet*, is the time between the **first insertion command** is issued and a packet has been observed for

the last of the (newly) inserted rules. For each configuration, we run the experiment 100 times and Figure 3.5 shows the median result as well as the 10^{th} and 90^{th} percentiles, although the variations are small and cannot be easily viewed.

From Figure 3.5, we observe that even though the *transmission delay* for sending flow insertion commands increases with their number, this time is negligible when compared with data plane measurements (*first packet*). Notably, the *barrier notification* measurements are almost constant, increasing only as the transmission delay increases (difficult to discern on the log-log plot) and, critically, this operation returns before any *first packet* measurement. This implies that the way the *barrier notification* is implemented does not reflect the time when the hardware flow-table has been updated.

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

Flow insertion types

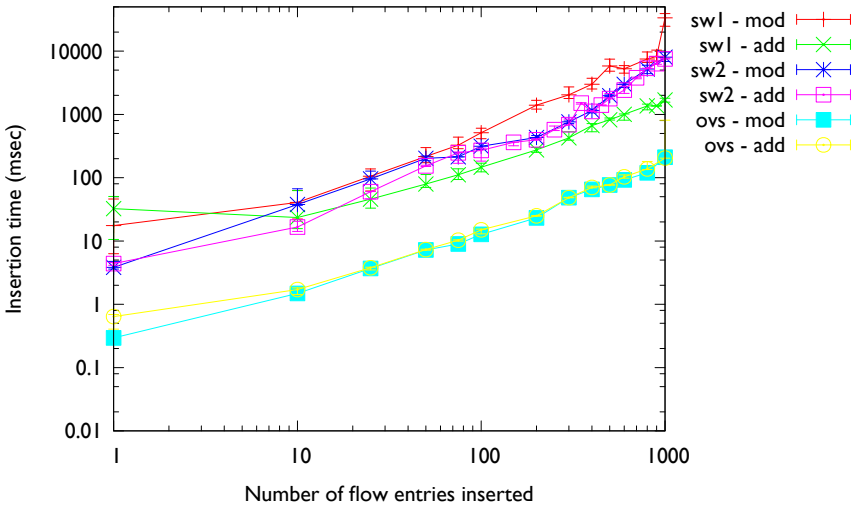


Figure 3.6: Delay of flow insertion and flow modification, as observed from the data plane (log-log scale).

We now distinguish between flow insertions and the modification of existing flows. With OpenFlow, a flow rule may perform exact packet matches or use wild-cards to match a range of values. Figure 3.6 compares the flow insertion delay as a function of the number of inserted entries. This is done for the insertion of new entries and for the modification of existing entries.

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

Clearly, the results depend both on the entry type and implementation. For example, exact match entries may be handled through a hardware or software hash table. Whereas, wild-carded entries, requiring support for variable length lookup, must be handled by specialized memory modules, such as a TCAM. With such possible choices and range of different experiments, the flow insertion times reported in Figure 3.6 are not generalizable, but rather depend on the type of insertion entry and implementation.

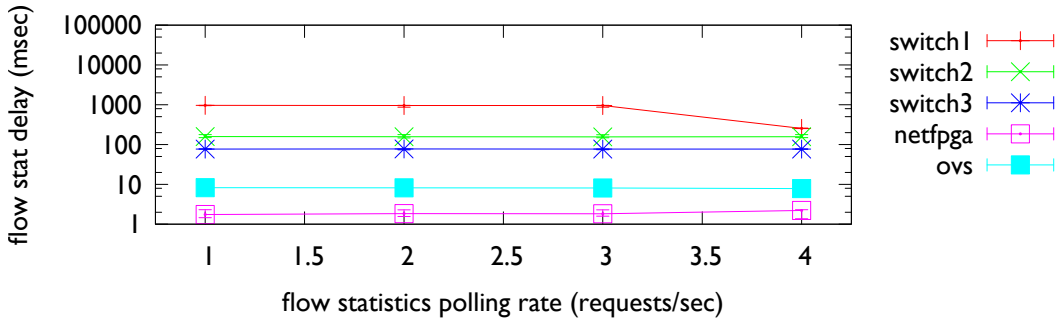
3.5.4 Flow monitoring

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

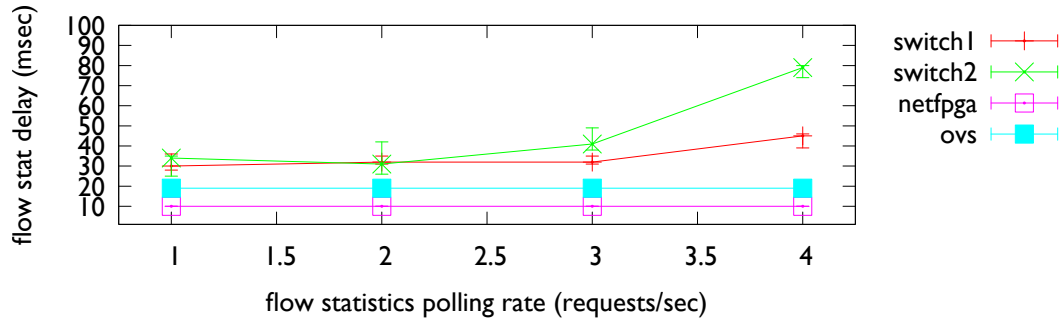
We now test the performance implications of the traffic statistics reporting mechanism of OpenFlow. Using OFLOPS, we install flow entries that match packets sent on the data path. Simultaneously, we start sending flow statistics requests to the switch. Throughout the experiment we record the delay getting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

timestamps of the probe packets.

Figure 3.7(a) reports the time to receive a flow statistics reply for each switch, as a function of the request rate. Despite the rate of statistics requests being modest, quite high CPU utilization is recorded for even a few queries per second being sent. Figure 3.7(b) reports the switch-CPU utilization as a function of the flow statistics inter-request time. Statistics are retrieved using SNMP. Switch3 is excluded for lack of SNMP support. *add texttt for all OpenFlow messages*



(a) Reply time.



(b) CPU utilization.

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

From the flow statistics reply times, we observe that all switches have (near-)constant response delays: the delay itself relates to the type of switch. As expected, software switches have faster response times than hardware switches, reflecting the availability of the information in memory without the need to poll multiple hardware counters. These consistent response times also hide the behavior of the exclusively hardware switches whose CPU time increases proportionally with the rate of requests. We ob-

serve two types of behavior from the hardware switches: the switch has a high CPU utilization, answering flow-stats requests as fast as possible (Switch2), or the switch delays responses, avoiding over-loading its CPU (Switch1). Furthermore, for Switch1, we notice that the switch is applying a pacing mechanism on its replies. Specifically, at low polling rates the switch splits its answer across multiple TCP segments: each segment containing statistics for a single flow. As the probing rate increases, the switch will aggregate multiple flows into a single segment. This suggests that independent queuing mechanisms are used for handling flow statistics requests. Finally, neither 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).

3.5.5 OpenFlow command interaction

An advanced feature of the OpenFlow protocol is its ability to provide network control applications with, e.g., flow arrival notifications from the network, while simultaneously providing fine-grain control of the forwarding process. This permits applications to adapt in real time to the requirements and load of the network Handigol et al. [2009]; Yap et al. [2009]. Using OFLOPS advanced measurement instrumentation, we develop a test scenario of dynamic network control, in order to understand the behavior of the switch control and data plane. In this scenario, we emulate the simultaneous querying of traffic statistics and modification of the flow table. More specifically, we extend Section 3.5.3 by showing how the mechanisms of traffic statistics extraction and table manipulation interact. Specifically, we initialize the flow table with 1024 exact match flows and measure the delay to update a subset of 100 flows. Simultaneously, the measurement module polls the switch for full table statistics at a constant rate. The experiment uses a constant rate 10Mbps packet probe to monitor the data path, and polls every 10 seconds for SNMP CPU values.

In this experiment, we control the probing rate for the flow statistics extraction mechanism, and we plot the time necessary before the modified flows become active in the flow table. For each probing rate, we repeat the experiment 50 times, plotting the median, 10^{th} and 90^{th} percentile. In Figure 3.8 we can see that, for lower polling rates, implementations have a near-constant insertion delay comparable to the results of Section 3.5.3. For higher probing rates on the other hand, Switch1 and Switch3 do

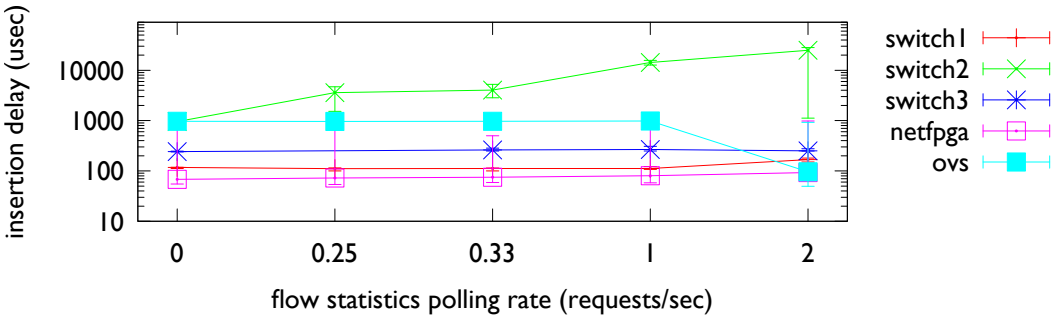


Figure 3.8: Delay when updating flow table while the controller polls for statistics.

not differ much in their behavior. In contrast, Switch2 exhibits a noteworthy increase in the insertion delay. This is explained by the CPU utilization increase incurred by the flow statistics polling (Figure 3.7(b)). Finally, Open vSwitch exhibits a marginal decrease in the median insertion delay and at the same time an increase in its variance. We believe this behavior is caused by interactions with the OS scheduling mechanism: the constant polling causes frequent interrupts for the user-space daemon of the switch, which leads to a batched handling of requests.

3.6 OpenFlow Macro-experimentation

OFLOPS, along with Cbench Cbench [2010], provide a sufficient set of tools to profile functionality of OpenFlow building blocks and measure low-level capabilities. Nonetheless, in order to understand the impact of control scalability in computer network, we require mechanisms to transform such models in network performance metrics, under specific traffic models and network topology. A computer network comprises of multiple functional units with interrelated functionality and multiple performance tuning parameters(e.g. Bufferbloat problem Gettys and Nichols [2011] highlights that multiple network buffers can affect flow performance during congestion). Developing analytical models that can provide high accuracy estimations is highly complex and the large number of network parameters make such models intractable. In order to reason about performance and correctness, developers have to revert to an experimental approach. In the related literature on network evaluation there have been

two main experimental mechanisms: *realistic-testbed* and *simulation*.

Realistic testbeds reconstruct in full detail the properties of the deployment environment. They provide an optimal measurement environment with complete control over the parameter of the experiment and excellent time scalability. Nonetheless, realistic testbeds incur significant resource and configuration overhead, which scale badly as the experiment size increases. Setting up a realistic testbed for datacenter networking requires a large number of machines and network devices with identical functionality in respect to the deployment environment, interconnection planning and careful metrication and analysis of the resulting system. In an effort to improve resource scalability for realistic testbeds, the research community has established a number of shared testbeds. Shared testbeds employ techniques such as virtualization and statistical multiplexing, and provide low-level user access to sizable infrastructures Emulab [2000]; PlanetLab [2007]. However, shared testbeds are not always a good fit for network experiments. In such testing environments, there is limited resource control, while resource sharing may introduce measurement noise, which is not always detectable and removable.

In the simulation approach, researchers replace parts of the functionality of the system with simplified models Issariyakul [2012]; Varga and Hornig [2008]. Simulation reduces the complexity of large scale network experiments, and provides resource scalability. Nonetheless, the scaling property has inherent limitations. Firstly, the fidelity of the results depends greatly on the validity of the model assumptions. Secondly, in order to simulate network experiments, users usually need to readjust the logic of their experiments in order to fit the abstraction of the underlying models. For example, POSIX socket-based controllers need to modify the control channel abstraction in order to match the API of the simulation platform, while forwarding plane traffic may have to transform in a stochastic model.

SDNSIM[1] is a novel network experimentation framework, that bridges the two aforthmentioned approaches. The framework is written in OcaML, a high performance functional language, and extends the functionality of the Mirage [2] library OS. Developers can describe the desired network functionality over the Mirage OS abstraction,

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

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

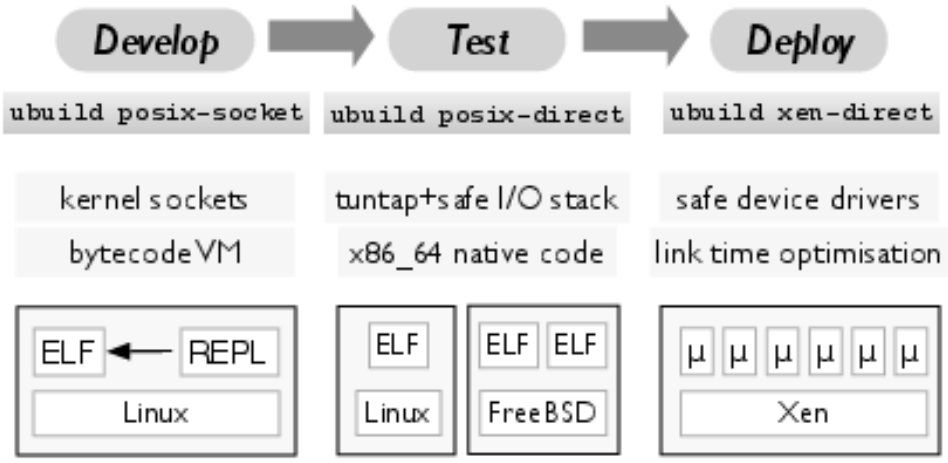


Figure 3.9: Specialising a Mirage application through recompilation alone, from interactive UNIX Read-Eval-Print Loop, to reduce dependency on the host kernel, and finally a unikernel VM.

and at compilation transform the experiment definition in a concrete experiment realisation. SDNSIM provides two experimentation options: *Simulation*, transforms the experiment definition in ns-3 Henderson et al. [2006] simulation, and *Emulation*, translates the experiment definition in Xen-based emulation of the experiment.

3.7 Mirage Library OS

todowhy do I use Mirage?

- Simple abstraction that can translate to any backend.
- Provides all the required functionality.
- Network layer doesn't incur complexity that can change logic.
- very low memory CPU requirement by the platform.

SDNSIM

Mirage is a cloud service development framework written in OCaml. Mirage applications are single purpose appliances that are compile-time specialised into standalone kernels deployable on the Xen platform. The aim of the framework is to provide small

size cloud OS images that are efficient and secure. In order to achieve this, Mirage revisits the idea of library OS; OS functionality is separated into logical modules and added to an appliance only if the code expresses an explicit dependency. As a result, Mirage can generate small size VM images with very fast boot times. Furthermore, using OCaml, a type-safe functional language, the framework is able to mitigate a number of security attacks to applications.

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

In terms of concurrency, Mirage uses the Lwt cooperative threading library abstraction Lwt [2013]. Lwt provides an OCaml syntax extension that can annotate blocking IO and internally evaluate blocking functions into event descriptors to provide straight-line control ow for the developer. Written in pure OCaml, Lwt threads are heap-allocated values, with only the thread main loop requiring a C binding to poll for external events. Mirage provides an evaluator that uses Xen polling to listen for events and wake up lightweight threads. The VM is thus either executing OCaml code or blocked, with no internal preemption or asynchronous interrupts. The main thread repeatedly executes until it completes or throws an exception, and the domain subsequently shuts down with the VM exit code matching the thread return value. A useful consequence is that most scheduling and thread logic is contained in an application library, and can thus be modified by the developer as they see fit.

Mirage OS provides a simple API to applications developers, sufficient for systems programming. This functionality is implemented by two core modules, named *Net* and *OS*, which expose a minimum API to the network and the device management stack. The simplicity of the OS and Net modules, permit Mirage to compile code to other target backends, apart from the Xen platform. Specifically, Mirage can generate UNIX

binaries, using both the POSIX library network functionality and raw sockets, and even Javascript executables that run in a browser. There is also currently an effort to port Mirage in the FreeBSD kernel as well as over the BareMetalOS Return Infinity [2013], an assembly OS. The diverse set of deployment backends, provides a sufficient environment for test and optimization, as depicted in Figure 3.9. Developers build initially their core logic over the POSIX backend in order to test the correctness of the code, then they can try their code over the Mirage default network stack, to perform a small scale performance evaluation, and finally they can synthesize the resulting deployable Xen Image.

3.8 SDNSIM design

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

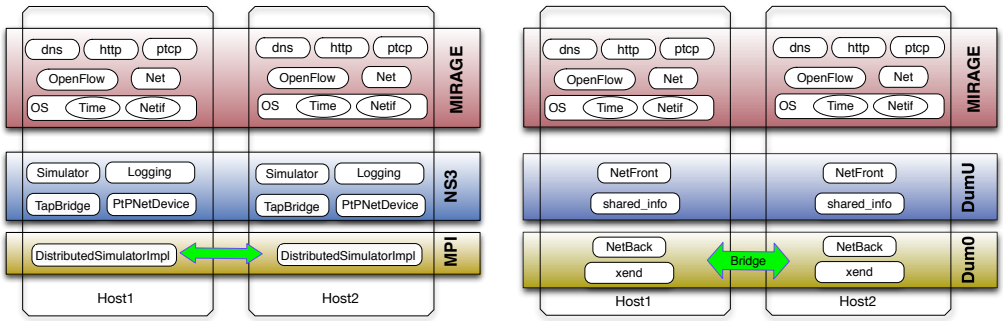
Listing 3.1: A sample SDNSIM configuration file interconnecting a server and a client host

From the developer perspective, SDNSIM consists of a single executable that functions as an OCaml build system. Developers implement host functionality as Mirage applications, and use a single XML file to describe the network topology and assign functionality to network nodes. A sample XML file is presented in Listing 3.1. The configuration describes a simple client (host1) - server (host2) configuration. A minimum experiment definition must define the core code module (topology@module), the target executable (topology@backend) and the duration of the experiment (topology@duration). In order to define a host, SDNSIM uses a host XML entity. Each host entity must define a host name (node@name) and a host main function (node@main), while a number of named parameters (node/param) can be passed to the main function. Finally, the link XML entity defines a link between two hosts (link@src, link@dst) along with the device (link@queue_size, link@pcap) and propagation properties (link@rate, link@delay), using the link XML entity. Links can also be used to integrate external network interfaces into the simulation, in order to allow the experiment to interact with entities outside of the experiment scope.

The functionality of a node in the SDNSIM platform can be split in 3 layers. A simple representation of the architecture of an SDNSIM host is depicted in Figure 3.10. The top layer contains the application logic of the host. This layer is defined by the developer and encodes the traffic model and the network forwarding logic of the experiment. In order to allow realistic traffic patterns, SDNSIM can reuse all the application protocol libraries supported in the Mirage platform, like DNS, HTTP, SSH and OpenFlow. In addition, we modify the OpenFlow processing code and introduce a latency control mechanism. Using the rate control mechanism, SDNSIM can simulate and emulate switch and controller models, measured through the OFLOPS and Cbench tools. We also have re-implemented in OCaml the pttcp tcp test tool Pratt [2002] for model driven traffic generation.

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

of SDNSIM experiment with each execution environment, as well as, the employed mechanisms to represent . Since SDNSIM aims for high network accuracy,we focus our presentation on how integration achieves network and time fidelity.



(a) ns-3 (b) Xen

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

3.8.1 Xen backend

A Mirage Xen Image uses a simple PV Boot mechanism. The mechanism initializes a VM with a single virtual CPU, loads the Xen event channel and jumps to the main function of the application which runs the thread scheduling loop. Mirage threads use the Lwt language extension, an event-driven asynchronous programming library. Using Lwt syntax, a closure can be noted as blocking, and spawn a new Lwt thread. An Lwt thread is either executing code or idles waiting for an IO event or a synchonisation primitive (e.g. a semaphore) or a timer timeout. Mirage threads are cooperative and non-preemptive and the main scheduling loop works as follow: Execute any resumable sleeping threads, calculate the time left until the next timer event and poll for events from the event channel until that time. Timing integration with the Xen platform is achieved through the Xen shared_info struct, a structure containing CPU time information and shared between the Dum0 and the DumU VMs. Network integration is implemented over Xen NetFront driver and IO polling is integrated with the Xen event channel.

SDNSIM uses the Xen Managment API (XAPI) Xen Project to setup an emulation over the XEN platform. SDNSIM is able to create, start and stop VM instances and

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

3.8.2 ns-3 backend

ns-3 Henderson et al. [2006] is a discrete-time packet-driven network simulation framework. The core of the system consists of a discrete time simulation engine, while a set of ns-3 libraries provide an extensive set of network applications, routing protocols, data-link layer protocols and link emulations models. ns-3 is fundamentally an extensive refactor of the ns-2 code-base, which aims to provide a stable core of simulation functionality and models.

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

of current network datacenters.

Finally, in order to increase the scalability of the SDNSIM simulation backend, we use a distributed version of the ns-3 simulation engine. The simulation engine spawns a different process for each host of the simulation and an MPI-based synchronisation algorithm establishes conservative clock synchronisation and distributed event execution Pelkey and Riley [2011].

3.9 SDNSIM evaluation

We evaluate performance and precision of the SDNSIM platform using small scale micro-benchmarks that target the performance of the OpenFlow protocol library, as well as, the scalability of the ns-3 backend. In Madhavapeddy et al. [2013], there is an exhaustive analysis of the performance of the Mirage platform, which we omit from this section. In the rest of this section we compare the performance of our Mirage OpenFlow controller (Section 3.9.1) and Mirage OpenFlow switch (Section 3.9.2) with existing equivalent software packages and evaluate the scalability of the ns-3 backend (Section 3.9.3).

3.9.1 Mirage Controller Emulation

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

Our benchmark setup uses the *Cbench* Cbench [2010] measurement tool. Cbench emulates multiple switches which simultaneously generates `pkt_in` messages. The program measures the processing throughput of each controller. It provides two modes of operation, both measured in terms of `pkt_in` requests processed per second: *la-*

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

Table 3.3: OpenFlow controller performance.

tency, where only a single `pkt_in` message is allowed in flight from each switch; and *throughput*, where each emulated switch maintains a full 64 kB buffer of outgoing 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 from switches.

We emulate 16 switches concurrently connected to the controller, each serving 100 distinct MAC addresses. We run our experiments on a 16-core AMD server running Debian Wheezy with 40 GB of RAM and each controller configured to use a single thread of execution. We restrict our analysis to the single-threaded case as Mirage, at time of testing, does not support multi-threading. For each controller we run the experiment for 120 seconds and measure the per-second rate of successful interactions. Table 3.3 reports the average and standard deviation of requests serviced per second.

Unsurprisingly, due to mature, highly optimised code, *N*OX fast shows the highest performance for both experiments. However, the controller exhibits extreme short-term unfairness in the throughput test. *N*OX provides greater fairness in the throughput test, at the cost of significantly reduced performance. Maestro performs as well as NOX for throughput but significantly worse for latency, probably due to the overheads of the Java VM. Finally, Mirage throughput is somewhat reduced from NOX fast but substantially better than both NOX and Maestro. In addition, the Mirage controller achieves the best product of performance and fairness among all tested controllers in the throughput test. Comparing latency, Mirage performs much better than Maestro but suffer somewhat in comparison to NOX. From the comparison results, we conclude that the Mirage controller performance is comparable to the performance of existing controlling platforms and our emulation environment can reproduce the performance of realistic OpenFlow deployment.

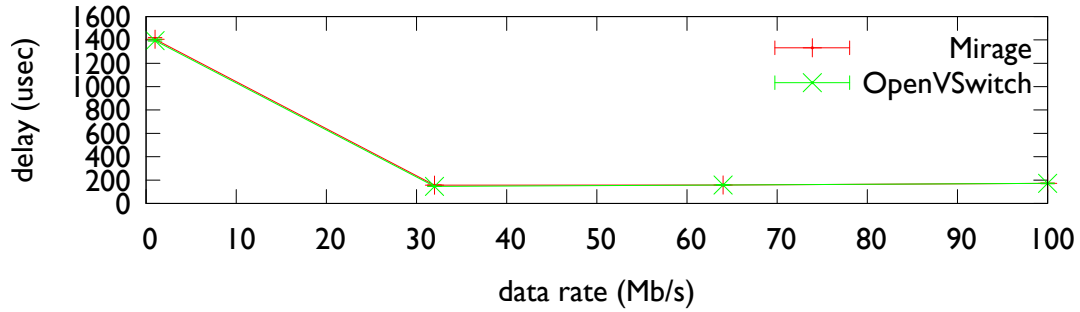


Figure 3.11: Min/max/median delay switching 100 byte packets when running the Mirage switch and Open vSwitch kernel module as domU virtual machines.

3.9.2 Mirage Switch

We benchmark our Mirage OpenFlow switch implementation through a baseline comparison with the Open vSwitch (OVS) Pettit et al. [2010] kernel implementation. We develop, using the OFLOPS framework, a simple forwarding test and measure the switching latency incurred by each implementation. For this experiment we use two virtual machines, one running the OFLOPS code, the other running the OpenFlow switch configured with three interfaces bridged separately in dom0. One interface provides a control channel for the switch, while the other two are used as the switch's data channels. Using OFLOPS, we generate packets on one of the data channels and receive traffic on the other, having inserted appropriate flow table entries prior to the beginning of the test. We run the test for 30 seconds, a sufficient measurement period to detect statistically significant results. We use small packets (100 bytes)[1] and vary the data rate.

Figure 3.11 plots as error boxes the min, median and max of the median processing latency of ten test runs of the experiment. We can see that the Mirage switch's forwarding performance is very close to that of the Open vSwitch, even mirroring the high per-packet processing latency with a probe rate of 1 Mb/s; we believe this is due to a performance artefact of the underlying dom0 network stack. We omit packet loss, but can report that both implementations suffer similar levels of packet loss. From the comparison results of the Mirage OpenFlow switch, we can conclude, that our OpenFlow

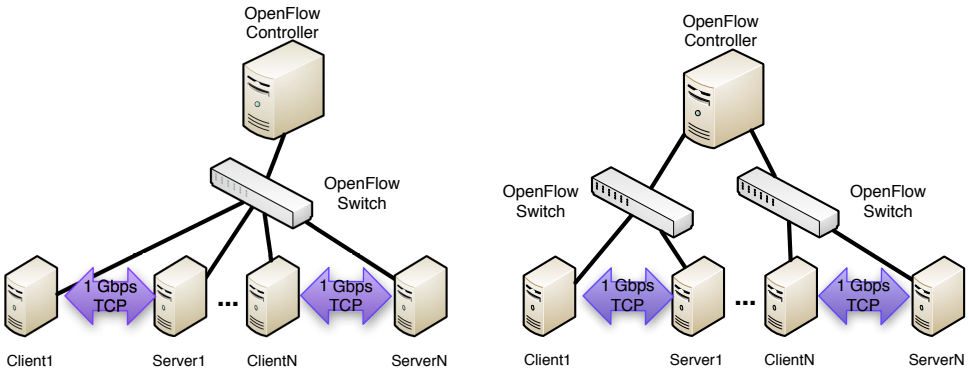
[1] we use a packet size slightly larger that the minimum packet size because we append in the payload packet generation information (e.g. packet ID, packet generation timestamps).

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

Table 3.4: Wall-clock delay to run a 30 seconds simulation time of the topology presented in Figure 3.12, for variable number of network hosts. SDNSIM simulation of a network experiment using the ns-3 backend scales linearly as the traffic is localised.

switch can emulate software switch functionality. Nonetheless, the minimum latency of our switch emulation (approximately 100 msec) is 2 order of magnitude higher than a hardware switch (approximately 0.5-1 μsec). In order to bridge this latency gap, we propose the introduction of a slowdown factor in the emulation. A slowdown factor of ten will reduce by ten times the time reported by the clock abstraction of the Mirage platform and by ten the link rate allocated to each VIF. As a result, the slowdown factor can provide a semantically correct slowdown to the time of the experiment and use more efficiently the computational resources of the server.

3.9.3 ns-3 performance



(a) centralised topology (b) distributed topology

Figure 3.12: Network topology to test the scalability of ns-3-based simulation. We simulate two topologies: a centralised topology (Figure 3.12(a)) and a distributed topology (Figure 3.12(b))

In order to test the scaling properties of the ns-3 backend we perform a simple topology experiment, depicted in Figure 3.12. The topology consists of a number of switches and an even number of hosts, split in pairs, connected through a 1 Gbps links. Each pair of hosts generates steady state TCP traffic at line rate. We use two variations of the topology: A *centralised topology* with all the pairs of hosts connected to a single switch (Figure 3.12(a)), and a *distributed topology*, with pairs of hosts distributed between two switches and traffic remaining local to the switch (Figure 3.12(b)). Each switch is connected to an OpenFlow controller that implements a learning switch. The experiment executes 30 seconds of simulation time on a 24-core AMD server with 128 GB of RAM.

In Table 3.4, we present the wall-clock execution time and the slowdown factor of each simulation. From the results we note that the real time to execute a simulation in SDNSIM depends to a great extend on the number of network events; as we increase the number of host and, consequently, the number of packets, the execution time increased. An additional insight from the results of the experiment occurs from the comparison between the centralised and distributed topology. Due to the distributed simulation engine, the performance of the simulation is highly benefited when network events are kept local, as the overall simulation task can be parallelized. In the distributed topology, network events are distributed between the two switch and they execute independently.

3.10 Security Tradeoffs on Datacenter Network Micro-control

Maybe remove this section

3.11 Summary

This chapter discusses the scalability of the main SDN approach

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 4.6 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 authorised member of the household, owner or not, may be able to exercise control based on specifics of the local context.

4.1 Technological and Social aspects of home networking

The growth of IP enabled devices over the last decade also means many households are now exploring the use of in-home wired and wireless networking, not only to allow multiple computers to share an Internet connection but also to enable local media sharing, gaming, and other applications. In computer network literature, a number of studies have been published that highlight the distinct properties of the home network environment. In Subsection 4.1.1 we present briefly the connectivity and traffic mix characteristics of home networks based on the outcome of relevant studies, while in Subsection 4.1.2 we discuss the relation of home networking technologies with the social context of the home, based on relevant ethnographic studies. Home networking is in parallel a social and a technical system. These two aspects interrelate dynamically and create an interesting feedback loop.

4.1.1 Home Networking as a system

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

Recent work on home network characterisation highlights high variability and a significant number of connected devices per household. In Cioccio et al. [2011] the authors develop the Homenet measurement framework, an end-host active measurement tool accompanied by a user survey. The user survey reports that a home network

connects on average 7-8 device, while active probing discovers on average only 1-2 device active during measurement. Further, the user survey reports a wide range of connected devices (e.g. smartphones, tablets, game consoles, etc.). This points out that the home network hosts a wide-range of networked devices with diverse operating systems and network stacks, and any modification in network functionality *m*ust be backwards compatible. Additionally, in Hätönen et al. [2010] the authors test a number of off-the-self home routers and unveil high variability in the semantics of the NAT and DNS functionality implemented by the router firmware. Nonetheless, since such routers are widely deployed in home networks, users network behaviour is resilient to such variability.

Home networks use predominantly wired and wireless Ethernet technologies to establish connectivity. Modern PCs provide currently support for both technologies. Wired network adapters, in their majority, support 802.3ab (1 Gbps full-duplex) standard and wireless network adapters support predominantly 802.11a/g (54 Mbps) standards with hardware-accelerated encryption, while 802.11n (600 Mbps) standard gains popularity. Modern wired home network connectivity provides lossless layer-2 transmission, but is less flexible to accommodate spatial mobility within the household. Wireless technologies are more user friendly, but their performance is susceptible to environmental parameters. The Homenet measurement Cioccio et al. [2011] study reports than on the analysed sample wireless signal strength is on average pretty good ($<$ -80dBm). In addition, the wireless card of the measurement PC was able to receive on average 10 advertisements from nearby wireless networks, a third of which was using overlapping channel. Interestingly, these observations point out the evolution of wireless technology and contrast earlier results on Yarvis et al. [2005]. The multiple SSID receipt also highlights a significant access control problem for homeowners. A wireless network often reaches beyond the limits of a household. This problem is usually partially mitigated using WPA security encryption. Nonetheless, this policy is crude, as password knowledge provides full access to the home network and there isn't a user-friendly mechanism to revoke access.

Home network broadband connectivity has been analysed extensively by the research community. On a governmental level, official independent authorities engage in extensive ISP measurement to evaluate the service provided by broadband providers FCC; ofcom [2012]. In Dischinger et al. [2007a], the authors contact an active network mea-

surement of residential broadband connections. The results of the analysis present significant performance differences between ISPs and pinpoint the performance bottleneck on the last mile of the broadband link. In Sundaresan and de Donato [2011] the authors integrate various measurement mechanisms into the home router firmware and contact a long-term performance measurement. In their analysis they unveil various differences in ISP performance and measure the impact of network throttle mechanisms, like PowerBoost DSLreports. The paper points out that broadband performance has multiple factors and cannot be measurable by a single test, while critical performance factors are distributed across the network. Finally, Netalyzer Kreibich et al. [2010], a web-based java-applet evaluating network connectivity and protocol openness, provides useful insight on broadband latency. Netalyzer, among other tests, incorporates a network buffer measurement tool, which floods a network path with packets. The analysis of the measurement results revealed that network buffer in the ISP network induce latency of up to 200 msec.

Home networks also have distinct network traffic properties in comparison to other network environments. In Reggani et al. [2012] the authors study the network traffic from a host point of view for various network environment. In a home network setting, hosts generate significant traffic volumes of filesystem and P2P applications and a large portion of the traffic remains local and is observable only from within the network. In terms of the home network Internet traffic pattern, research has highlighted both significant variability and location-dependence. In Cho et al. [2006] authors describe that during the time of their analysis in Japan there was a significant usage of P2P applications. More recent studies Maier et al. [2009] point out that users in Germany have shifted interest towards web applications and a large portion of traffic is HTTP-based. Similar results are pointed out in Erman et al. [2011], where HTTP flows from an ISP trace are mapped to application types, and online video and one-click hosting services appear as the predominant application classes.

4.1.2 Home Network as a social activity

Home networking technologies have been an interesting domain of study and application for HCI, Ubiquitous computing and sociology, since it provides an excellent environment to study the interaction between users and technology. Studies in the

field usually engage in user interviews in order to understand how users perceive and interact with technology.

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

A number of user studies have augmented the factors that shape home networking adding as an important factor the design of the building within which the network is installed. In Rodden and Benford [2003] authors describe a 7-layers model devised by the American writer Stewart Brand in Brand [1995] which describes how homes evolve architecturally after their initial establishment. Using this model authors analyse the relationships between Ubquitous technologies and home design. This study is further focused on the home networking technologies in Chetty et al. [2007]. Authors contact a user study in order to understand how the home design relates to the choices of users regarding their home network. The study describes how the user network decisions are affected by the design of the house, e.g., location of the network router, while at the same time how the users confuse the limits of the house with the limits of

their network, e.g. users assume that encrypting their home network is not important since it is contained within the limits of the house. In Crabtree et al. [2003]; Rodden et al. [2004] the authors study a number of family homes and monitor the real time communications and the ways in which information is produced and consumer within the house. In their study they concluide that a lot of these activities have a location reference within the house, which is related to the involved member as well as the house planning, while activities can be synthesized as sequences into higher order activities. Additionally, using these observations, they propose a framework which can model such interactions.

Finally, in Shehan and Edwards [2007] authors analyze some common management and configuration problem in home networks and project them in the respective design decisions of the network systems. They present a weight of evidence that problems with home networking are not amenable to solution via a 'thin veneer' of user interface technology layered atop the existing architecture. Rather, they are *structural*, emerging from the mismatch between the stable 'end-to-end' nature of the Internet and the highly dynamic and evolving nature of domestic environments.

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

4.2 Motivations

4.2.1 Home Network: Use cases

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

In fact, the home network is quite different in nature to both core and enterprise networks. Existing studies Shehan and Edwards [2007]; Shehan-Poole et al. [2008];

Tolmie et al. [2007] suggest domestic networks tend to be relatively small in size with between 5 and 20 devices connected at a time. The infrastructure is predominately cooperatively self-managed by residents who are seldom expert in networking technology and, as this is not a professional activity, rarely motivated to become expert. A wide range of devices connect to the home network, including desktop PCs, games consoles, and a variety of mobile devices ranging from smartphones to digital cameras. Not only do these devices vary in capability, they are often owned and controlled by different household members.

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

Negotiating acceptable use. *William and Mary have a spare room which they let to a lodger, Roberto. They are not heavy network users and so, although they have a wireless network installed, they pay only for the lowest tier of service and they allow Roberto to make use of it. The lowest tier of service comes under an acceptable use policy that applies a monthly bandwidth cap. Since Roberto arrived from Chile they have exceeded their monthly cap on several occasions, causing them some inconvenience. They presume it is Roberto's network use causing this, but are unsure and do not want to cause offence by accusing him without evidence.*

Welcome visitors, unwelcome laptops. *Steve visits his friends Mike and Elisabeth for the weekend and brings his laptop and smartphone. Mike has installed several wireless access points throughout his home and has secured the network using MAC address filtering in addition to WPA2. To access the network, Steve must not only enter the WPA2 passphrase, but must also obtain the MAC addresses of his devices for Mike to enter on each wireless access point. Steve apologizes for the trouble this would cause and, rather than be a problem to his hosts, suggests he reads his email at a local cafe.*

Sharing the medium socially efficient. *Richard is the teenage son of Derek and has a great interest in Music, downloading a lot of music from the Internet. Derek works some times in the night from home using the Terminal Services provided by his company. Tension is created between them as Derek blames Richard downloading activity for his poor performing remote desktop application.*

In such ways, simple domestic activities have deep implications for infrastructures

that generate prohibitive technical overheads. In the first scenario, the problem is simply that the network's behaviour is opaque and difficult for normal users to inspect; in the second, the problems arise from the need to control access to the network and the technology details exposed by current mechanisms for doing so.

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

4.2.2 Home Networks: Revolution!

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

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

We exploit human understandings of the local network and the home to guide management of the supporting infrastructure Crabtree et al. [2003] by focusing on the home router not only as the boundary point in an edge network but as a physical device which

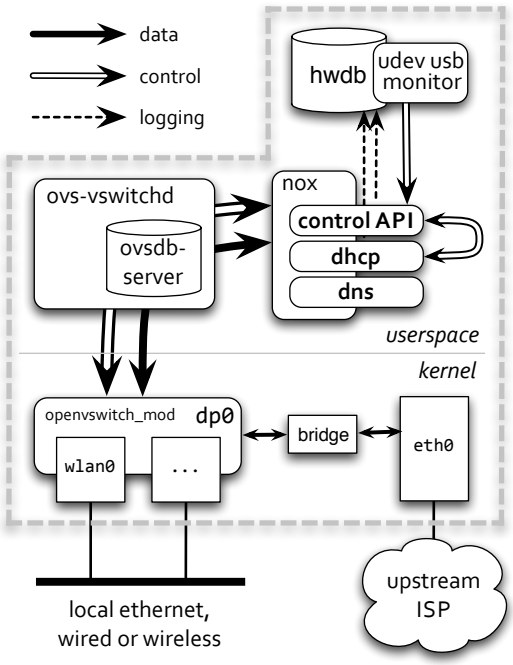


Figure 4.1: Home router architecture. Open vSwitch (*ovs**) and NOX manage the wireless interface. Three NOX modules provide a web services control API, a DHCP server with custom address allocation and lease management, and a DNS interceptor, all logging to the Homework Database (*hwdb*) (§4.4).

can be exploited as a point of management for the domestic infrastructure. Within our router, we focus on flow management for three reasons:

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

4.3 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

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

Open vSwitch OpenFlow implementation, a NOX controller exporting a web service interface to control custom modules that monitor and manage DHCP and DNS traffic, plus the Homework Database Sventek et al. [2011] providing an integrated network monitoring facility. This gives us a setup very similar to a standard operator-provided home router where a single box acts as wireless access point, multiplexes a wired connection for upstream connectivity to the ISP, and may provide a small number of other wired interfaces.

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

4.3.1 OpenFlow, Open vSwitch & NOX

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

Our network control logic, discussed in Section 4.4, is implemented over the NOX controller abstraction and split between 5 different modules. The C++ module *hwdb* synchronizes router state with the hwdb home database, presented in Section 4.3.2, the C++ module *homework_dhcp* implements our custom DHCP server, the C++ module *homework_routing* implements the forwarding logic of the design, the C++ module *homework_dns* implements the DNS interception functionality and the Python module *homework_rpc* exposes the control API as a Web service.

Our router provides flow-level control and management of traffic via a single OpenFlow datapath managing the wireless interface of the platform.[1] Control of the router

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

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

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

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

4.3.2 The Homework Database

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

The database is queried via a variant of CQL Arasu et al. [2005] able to express both temporal and relational operations on data, allowing applications such as our user interfaces to periodically query the ephemeral component for either raw events or information derived from them. Applications need not be collocated on the router as *hwdb* provides a lightweight, UDP-based RPC system that supports one-outstanding-packet semantics for each connection, fragmentation and reassembly of large buffers, optimization of ACKs for rapid request/response exchanges, and maintains liveness for

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

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

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

4.3.3 The Guest Board

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

- Wireless signal strength is mapped to device label transparency, so devices supplying weak signals fade into the background.
- Device bandwidth use is proportional to its label size, e.g., Tom's Laptop in Figure 4.2 is currently the dominant bandwidth user.

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

Devices in range appear on the screen in real-time, initially in the leftmost panel indicating they are within range of the home router but not connected. The central panel in the control displays machines actively seeking to associate to the access point. This zone exploits the underlying strategy of placing people in the protocol discussed in Section 4.4. When devices unknown to the network issue DHCP requests, the router's DHCP server informs the guest board and a corresponding label appears in this portion of the display. If a user wishes to give permission for the machine to join the network they drag the label to the right panel; to deny access, they drag the label to the left panel. The guest board provides both a central control point and, by drawing directly upon network information collected within our router, a network-centric view of the infrastructure.

4.4 Putting People in the Protocol

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

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

4.4.1 Address Management

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

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

We measured the performance of our DHCP implementation and found that, as expected, per-request service latency scales linearly with the number of simultaneous requests. Testing in a fairly extreme scenario, simultaneous arrival of 10 people each with 10 devices, gives a median per-host service time of 0.7s.

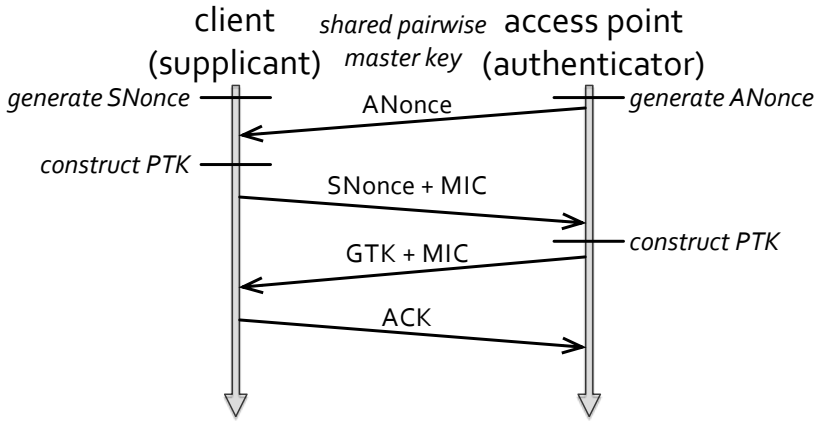


Figure 4.3: 802.11i handshake, part of the association process. Note that MIC (Message Integrity Code) is an alternate term for MAC, used in such contexts to avoid confusion with Media Access Control.

4.4.2 Per-Protocol Intervention

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

Our home router supports wireless Ethernet security via 802.11i with EAP-WPA2, depicted in Figure 4.3, using *hostapd*. In EAP-WPA2 security mechanism, the client (*supplicant*) and our router (*authenticator*) negotiate two keys derived from the shared master key via a four-way handshake, through the EAPOL protocol. The *Pairwise Transient Key* (PTK) is used to secure and authenticate communication between the client and the router; the *Group Transient Key* (GTK) is used by the router to broadcast/multicast traffic to all associated clients, and by the clients to decrypt that traffic. All non-broadcast communication between clients must therefore pass via the router at the link-layer (for decryption with the source's PTK and re-encryption with the destination's PTK), although the IP routing layers are oblivious to this if the two clients are on the same IP subnet.[1]

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

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

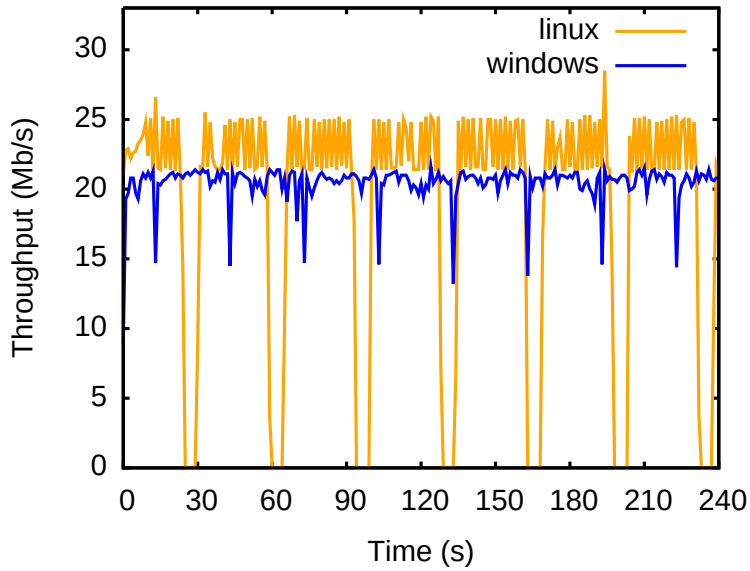


Figure 4.4: Affect on TCP throughput from rekeying every 30s for Linux 2.6.35 using a Broadcom card with the *athk9* module; and Windows 7 using a proprietary Intel driver and card.

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

To quantify the impact of 802.11i rekeying, we observed throughput over several rekeying intervals. Figure 4.4 shows the impact of setting the rekeying interval to 30s: rekeying causes a periodic dip in throughput as the wireless Ethernet transparently buffers packets during rekeying before transmitting them as if nothing had happened. This shows the trade-off between performance and responsiveness of this approach: to be highly responsive in detection of misbehaving clients imposes a small performance

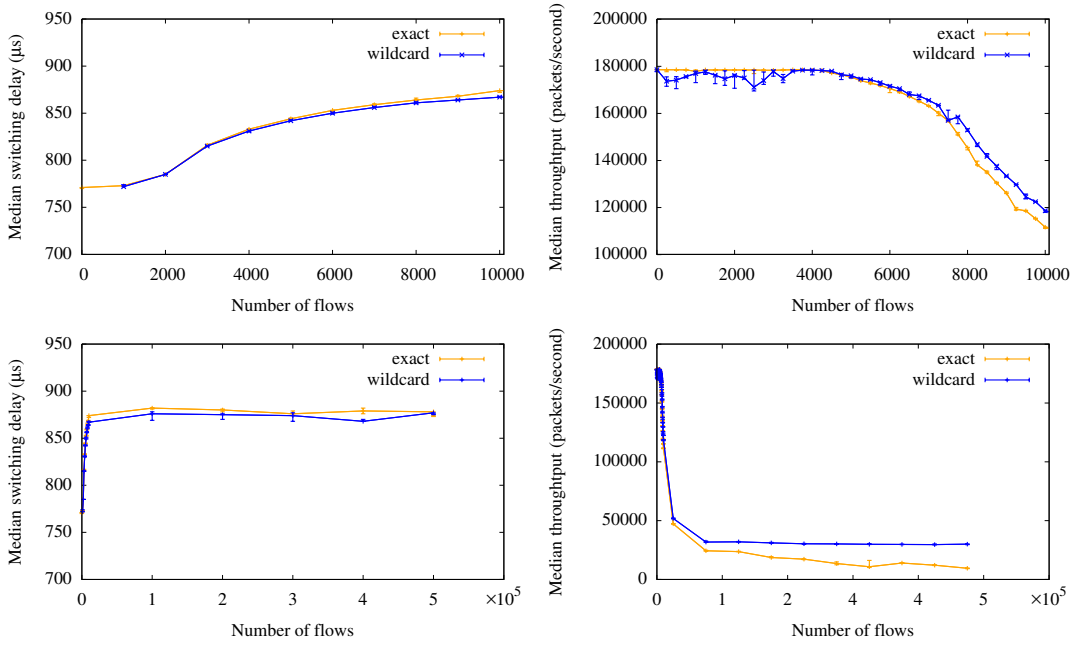


Figure 4.5: Switching performance of Open vSwitch component of our home router showing increasing per-packet latency (LHS) and decreasing packet throughput (RHS) with the number of flows. The lower set of graphs extends the x-axis from 10,000 to 500,000.

degradation. As a compromise, when a device is blacklisted, all of its traffic and subsequent rekeying exchanges are blocked. Thus the misbehaving device is prevented from sending or directly receiving any traffic before rekeying takes place, The device will be able to receive only broadcast traffic in the interim due, to the use of the GTK for such frames, until the AP initiate the negotiation of a new key. This allows us to pick a relatively long rekey interval (5 minutes) while still being able to respond quickly to misbehaving devices.

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

4.4.3 Forwarding

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

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

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

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

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

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

To assess switching performance, we examine both latency and packet throughput as we increase the number of flows, N, from 1–500,000. 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 (1500B) for the latency tests and minimal packets (70B) [2] for the throughput tests, selecting destinations at random on a per-packet basis. Results are presented as the median of 5 independent runs with error bars giving the min and max values.

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

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

Figure 4.5 shows median per-packet switching delay and per-flow packet throughput using either exact-match rules or a single wildcard rule per host. Performance is quite acceptable with a maximum switching delay of 560μs and minimum throughput of 40,000 packets/second; 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 accidental or malicious, our home router monitors the number of per-flow rules introduced for each host. If this exceeds a threshold then the host has its per-flow rules replaced with a single per-host rule, while the router simultaneously invokes user interface callback to inform the homeowner of the device's odd behaviour.

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

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

Table 4.2 shows the observed behaviour of a number of common home-networked devices: in short, all devices operated as expected once permitted access. DNS interception was not explicitly tested since, as an inherently unreliable protocol, all net-

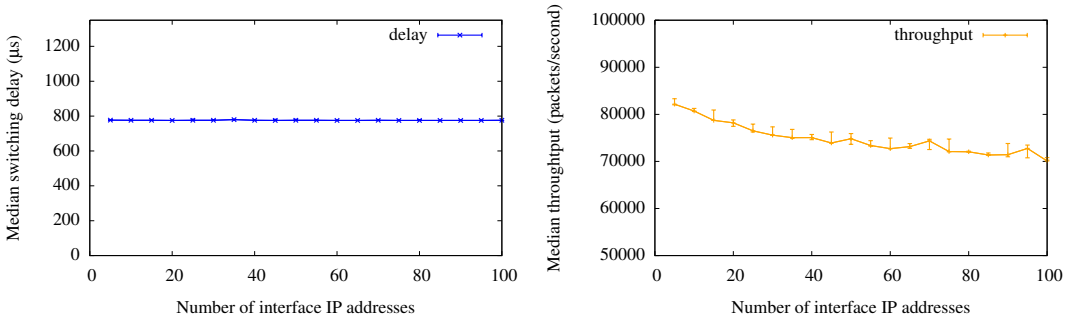


Figure 4.6: Switching performance of Linux network stack under our address allocation policy. Throughput (left figure) shows a small linear decrease while switching delay (right figure) remains approximately constant as the number of addresses allocated to the interface increases.

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.

working 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 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

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

0.1% (packets and bytes) in both homes.

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

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

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

In order to test the functionality of our approach, we set up a simple hardcoded version of our design. Specifically, we allocate a public IPv6 /64 prefix to our home router. The router uses the ISC DHCPv6 server implementation, configured to allocate addresses from a /120 subnet. Additionally on the router we run the RADVD daemon to reply appropriately to ICMPv6 messages. Using this network setup, we

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

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

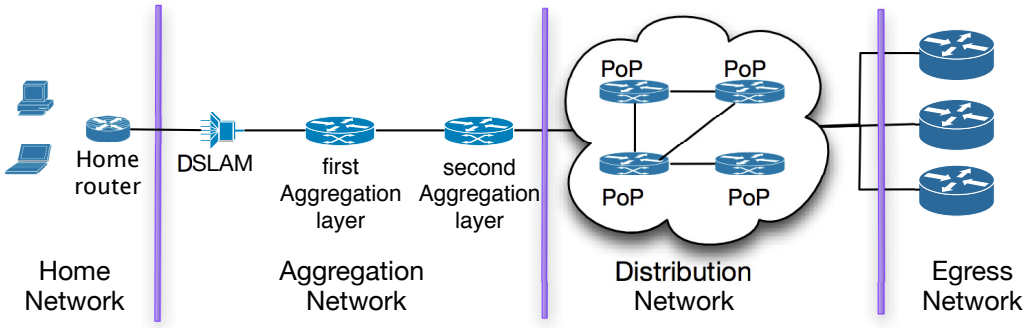


Figure 4.7: The path for each network packet of the home network to the Internet. ISP network can be split in 3 parts: The *Aggregation Network*, the *Distribution Network* and the *Egress Network*.

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

4.5 User-centric resource allocation

As we have discussed in Section 1.1, network application evolution has increased both the per-host resource requirements, as well as, the complexity of the resource allocation mechanism. Effectively, developing network architectures that can provide global application-specific network performance guarantees is a difficult task. The network community uses two common practices to tackle the resource allocation problem: network resource over-provision and application-aware traffic engineering. Resource over-provision increases available resources, in order to remove performance bottlenecks. Application-aware network engineering extends the network policy to manage efficiently traffic from specific applications, e.g. in an enterprise network VoIP devices are connected to a separate high-priority VLAN. In broadband networks these approaches have limited applicability. On one hand, resource over-provision cost is non scalable on the edges of the broadband network [1], while, on the other hand, application-aware network engineering raises legal issues for Internet providers Hahn and Wallsten

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

[2006]. In this section, we propose an architectural extension in the last-mile of the ISP, which enables homeowner to propagate per-application resource requirements. Using this mechanism, we scale the resource allocation problem in ISP management, while users can improve their network experience and gain higher understanding . In the rest of this section, we present the architecture of current broadband networks and discuss the inherent limitations in resource allocation (Section 4.5.1), we present a highly efficient network architecture for user-driven resource allocation (Section 4.5.2) and evaluate the impact of our approach on the performance of various traffic types (Section 4.5.3).

4.5.1 ISP resource allocation

In order to better introduce the reader to our system, we present in Figure 4.7 a visualisation of a broadband ISP network architecture. Although, exact network topology typically remains secret to each ISP, there is a high level design pattern. In this figure we present the network devices traversed by a packet sent from the home network to the Internet. In the Figure, we segment the network topology into four sections, the *home network*, *aggregation network*, *distribution network* and *egress network*. The aggregation network contains the Digital subscriber line access multiplexer (DSLAM), translating DSL packets to Ethernet format over the last mile, the Broadband Remote Access Server (BRAS), an authenticating and policy enforcing network device, and a number of layers of aggregation switches, multiplexing the traffic from the various BRAS onto the distribution network. Traffic from the aggregate network is routed through the distribution network towards the egress network of the ISP or another aggregation switch, if the packet is destined to an internal IP address. The distribution network consists of a small number of large routers interconnected using high capacity links. An ISP may typically use network tunnelling technologies, like MPLS, in the distribution network, in order to reduce forwarding information on the routers. The egress network of the ISP consist of large routers, hosted in AS IXP networks and connecting the ISP with peering ASes and the rest of the Internet.

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

tleneck can be explained by the high oversubscription ratio of such links. The oversubscription ratio commonly varies between the values 10:1 and 50:1 Program [2013], depending on the number of users connected to the DSLAM, but it is not standardised and is strongly affected by market dynamics Leach [2013].

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

Our architecture establishes a communication channel between the household and the ISP and enables user-driven dynamically resource control on the bottleneck link towards the aggregation network. More specifically, the ISP virtualises switch resources over the last-mile and delegates control over a subset of the available resources to a homeowner-managed OpenFlow controller. In the virtualized resources we additionally allocate minimum guaranteed bandwidth and a set of hierarchical priority queues for each household. The household controller, in real time, uses the OpenFlow protocol to forward high priority flows to the appropriate priority queue.

While the proposed solution is not able to provide a complete system to Internet-scale resource scheduling, we believe that it provides a sufficient mechanism to handle in a user-friendly manner congestion on the edge-network and scale the configuration complexity.

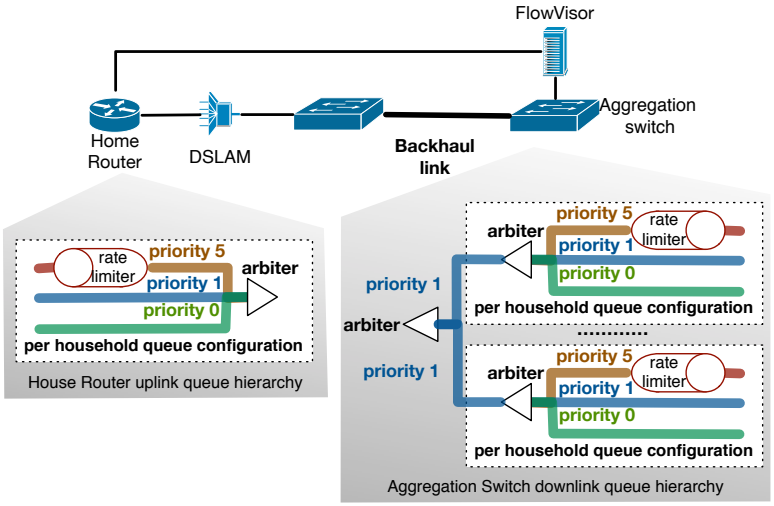


Figure 4.8: User-driven network resource allocation architecture. Switch handling the downlink from the backhaul network exposes a virtual slice to the homeowner through a FlowVisor instance. The switch allocates three queue primitives per-household: A *low latency, high priority* queue, a *medium priority*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

4.5.2 User - ISP communication

In the proposed architecture we split system functionality between three points in the network: a data-collection daemon on the end-systems of the home network, a policy enforcing daemon on the home network router and an OpenFlow controller to translate user performance requirements into ISP control policy.

End-system In order to map network flows to application, we develop a light cross-platform daemon running on home network hosts and recording connection table content. These information are collected from the connection table of the host network stack and stored in the HWDB database. The daemon accesses the connection table entries using the *netfilter-conntrack* netfilter.org project [2010] in Linux systems, the *Windows Filtering Platform* Microsoft [2012] in Windows systems and the *sysctl* for Darwin/MacOSX system. In order to match each network tuple with the respective applications, we use the `lsof` command in unix-like systems and the `netstat -p` command in Windows.

ISP infrastructure We present the network on the edge of the ISP network in Figure 4.8. In our user-centric resource allocation design the first layer of aggregation switches must expose OpenFlow control. The switch control plane is virtualised using the FlowVisor controller Sherwood et al. [2010]. Each home router is running an OpenFlow controller which connects to the FlowVisor service and exercises control over the home Internet traffic. Specifically, the ISP virtualises the OpenFlow forwarding table and exposes only a subset of the entries to each user. The FlowVisor instance propagates `pkt_in` messages to the home controller with a source or destination IP address equal to the home public IP address and the FlowVisor discards any *flow_mod* packet which does not contain an equivalent IP address. This FlowVisor configuration enforces a strong security mechanism; home controllers cannot control or eavesdrop traffic from other households.

At the aggregation switch and the home router we setup three priority queues for each household (The home router queues handle the upstream traffic of the household, while the aggregation switch queues handle the downstream traffic of the household): A *low latency, high priority* (LLHP) queue, a *medium priority* (MP) queue and a default queue. The LLHP queue has the highest priority between the household queues and uses a leaky bucket mechanism to rate limit traffic to the minimum guaranteed bandwidth per-household. This queue can be used by low-bandwidth low-latency applications. The HP queue doesn't enforce any rate limit, but it has a priority which is between the LLHP and the default queue. HP queue can be used by latency-tolerant high-priority applications. Finally, the default queue is used to handle all other traffic types. The queue hierarchies from each household are multiplexed towards the output port with equal priorities. Large scale queue hierarchies are currently supported by service provider grade network equipment (e.g. Cisco ASR 9000 router provide 500k queues per line card Cisco [2012a]) and ISPs use this functionality to implement their capping policy. By default, all network flows are forwarded to the default queue and the home OpenFlow controller at run-time can assign flows to higher priority queues based on the user configured policy. This resource allocation mechanism is constraint per-household by the number of flows that the FlowVisor exposes to each home. This network design can be extended to evolve further the ISP broadband economical model; Users can enhance network performance by purchasing additional flow table entries and increase minimum guaranteed bandwidth on the edge.

Because our system design provides extensive forwarding control to end-users, we fortify the design with a set of mechanisms to reduce the ability of end-users to compromise network functionality. Firstly, fair allocation of flow table entries to each household, ensures fair utilisation of the forwarding resources in the aggregate switch. Secondly, the FlowVisor instance rate limits OpenFlow control channel interactions per household to mitigate DoS attacks. Finally, the Flowvisor instance discards flows that may create loops in the network, e.g. `flow_mod` messages with an Output action that forwards packet to the incoming port.

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

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

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

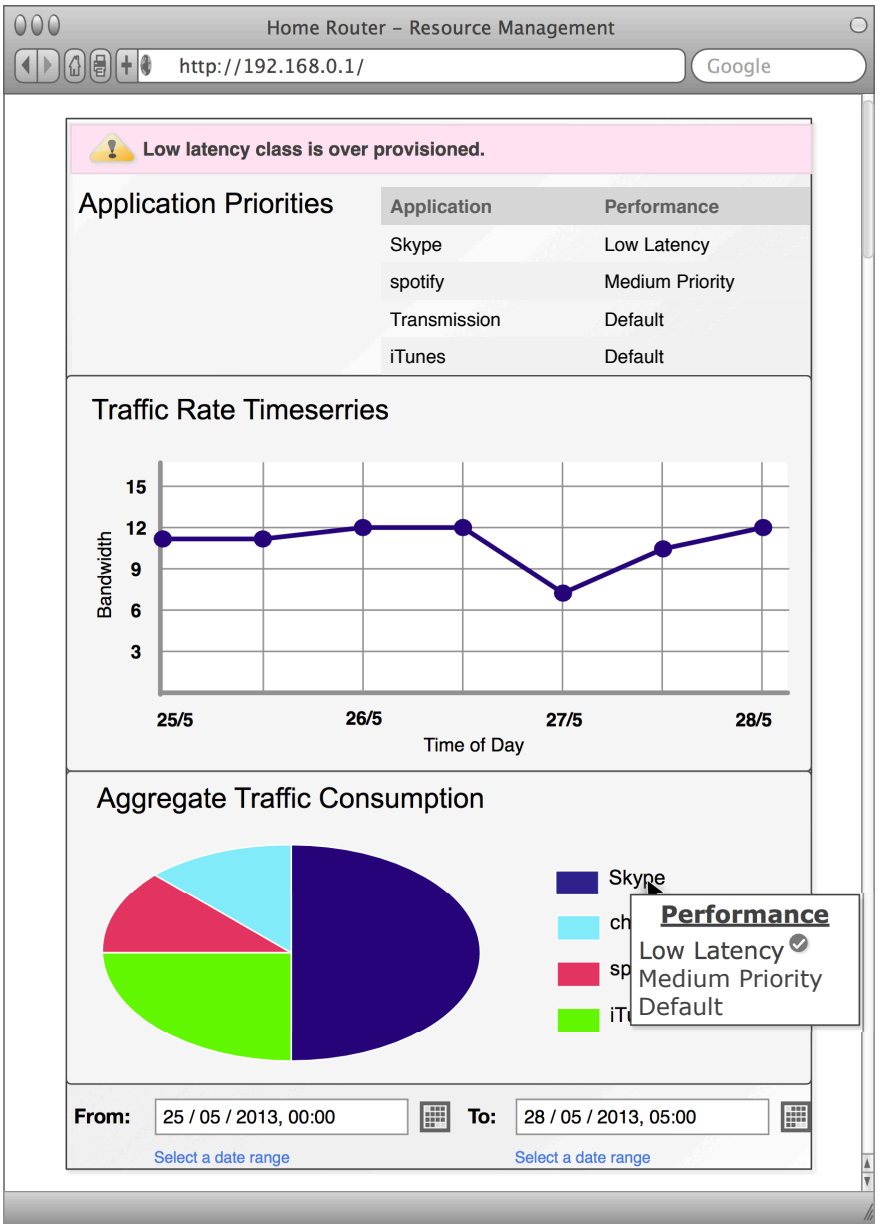


Figure 4.9: Performance mechanism user control. The user is able to view the network utilisation per application, as well as express application prioritisation.

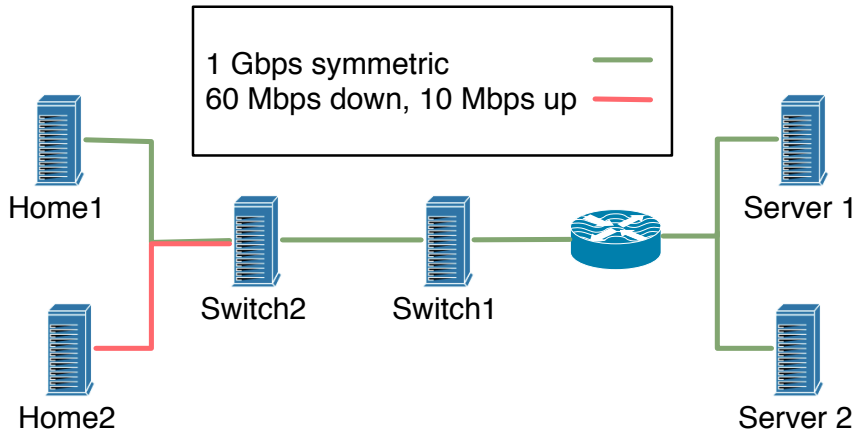


Figure 4.10: Experimental setup to evaluate the proposed resource management system.

maps the flow to an application and to a priority queue, through the HWDB database. The controller sends a `flow_mod` message with an `enqueue` action to assign the flow to the appropriate queue. In addition, if the application priority is not the default, then the controller will also send a `flow_mod` packet to the FlowVisor instance, to setup the incoming direction of the flow.

4.5.3 Evaluation

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

4.6 Conclusions

This paper has drawn upon previous user studies to reflect on the distinctive nature of home networks and the implications for domestic network infrastructures. Two particular user needs that arose from these studies were for richer visibility into and greater control over the home wireless network, as part of the everyday management of the home by inhabitants. We considered how to exploit the nature of the home network to shape how it is presented and opened to user control.

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

We use the Open vSwitch and NOX platforms to provide flow-based management of the home network. As part of this flow-based management, we exploit the social conventions in the home to manage introduction of devices to the network, and their subsequent access to each other and Internet hosted services. This required modification of three standard protocols, DHCP, EAPOL and DNS, albeit in their behaviour only *not* their wire formats, due to the need to retain compatibility with legacy deployed stacks.

Our exploration suggests that, just as with other edge networks, existing presumptions could usefully be re-examined to see if they still apply in this context. *Do we wish to maintain net neutrality in the home?* Inhabitants do not appear to see network traffic as equal, often desiring imbalance in performance received by different forms of traffic. *Must the end-to-end argument apply?* Householders understand and exploit the physical nature of their home and use trust boundaries to manage access; we have exploited these resources to explicitly manage the network. *Should communication infrastructures remain separate from the devices that use them?* In the home setting this separation proves problematic as people, ranging from the home tinkerer to the DIY expert, wish to interact directly with the network as they do with other parts of their homes' physical infrastructures. Our exploration suggests use of a range of displays

and devices existing not as clients exploiting the infrastructure but as extensions of the infrastructure making it more available and controllable.

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

Chapter 5

Scalable User-centric cloud networking

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

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

In this chapter, we present in Section 5.1 the motivation for this work, followed then by the key observations for our design in Section 5.2. In Section 5.3, we present the architecture of our strawman implementation and its integration with existing software. Finally, in Section 5.4 we present a number of micro-benchmark tests for our system

and conclude in Section5.5.

5.1 Personal Clouds

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

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

Section, we discuss the problems arising in Personal Cloud functionality over the Internet (Subsection 5.1.1), present existing approaches to the problem (Subsection 5.1.2) and introduce readers to the Signpost framework (Subsection 5.1.3).

5.1.1 Challenges

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

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

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

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

5.1.2 Approaches

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

Decentralised Personal Cloud :

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

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

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

cloud-assisted Personal Cloud :

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

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

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

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

In Wittie et al. [2010], authors report a significant impact on the 95th quantile network performance of cloud services, due to latency and packet losses incurred by the Internet. *Add dropbox measurement study*

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

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

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

5.1.3 Reconnecting the Internet

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

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

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

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

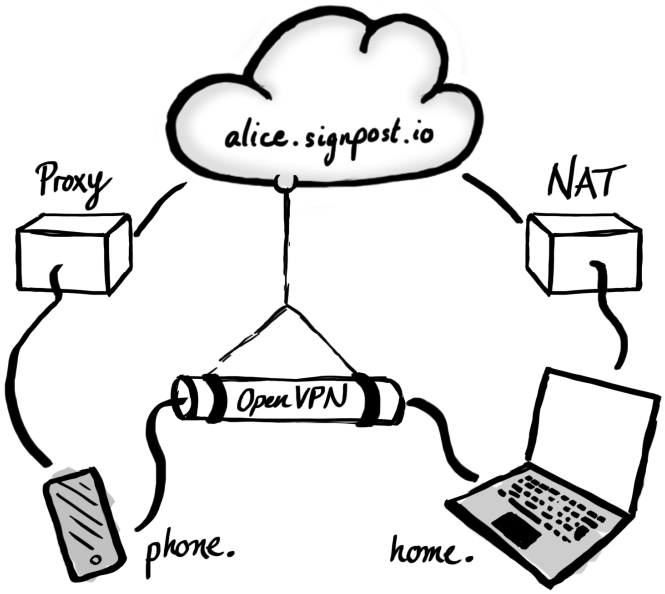


Figure 5.1: A simple example of the Signpost abstraction when the user Alice interconnects a smarthphone with the home computer over the Internet.

5.2 Signpost Design Goals

5.3 Signpost Architecture

Signpost is an Internet-wide secure inter-device communication system. The system reuses existing Internet protocols and connectivity mechanisms and provides an overlay local network. In Figure 5.2 we present a diagram of the Signpost architecture over different abstractions. In the lower section of Figure 5.2, we present the design of the Signpost software and its integration with existing applications. Signpost logic is contained in a single executable, and requires from the guest OS to expose an OpenFlow switch interface and redirect DNS queries to the embedded DNS resolver. The software consists of three main subsystems: A *Connection Engine* (Subsection 5.3.3) that sets up and manages *Network Tactics* (Subsection 5.3.1 in order to establish end-to-end paths, a local DNS resolver and an OpenFlow-based *Signpost router* (Subsection 5.3.2) that enhances the normal OS routing functionality with Signpost logic. In the middle layer of Figure 5.2, we present the control plane interconnection of Signpost devices.

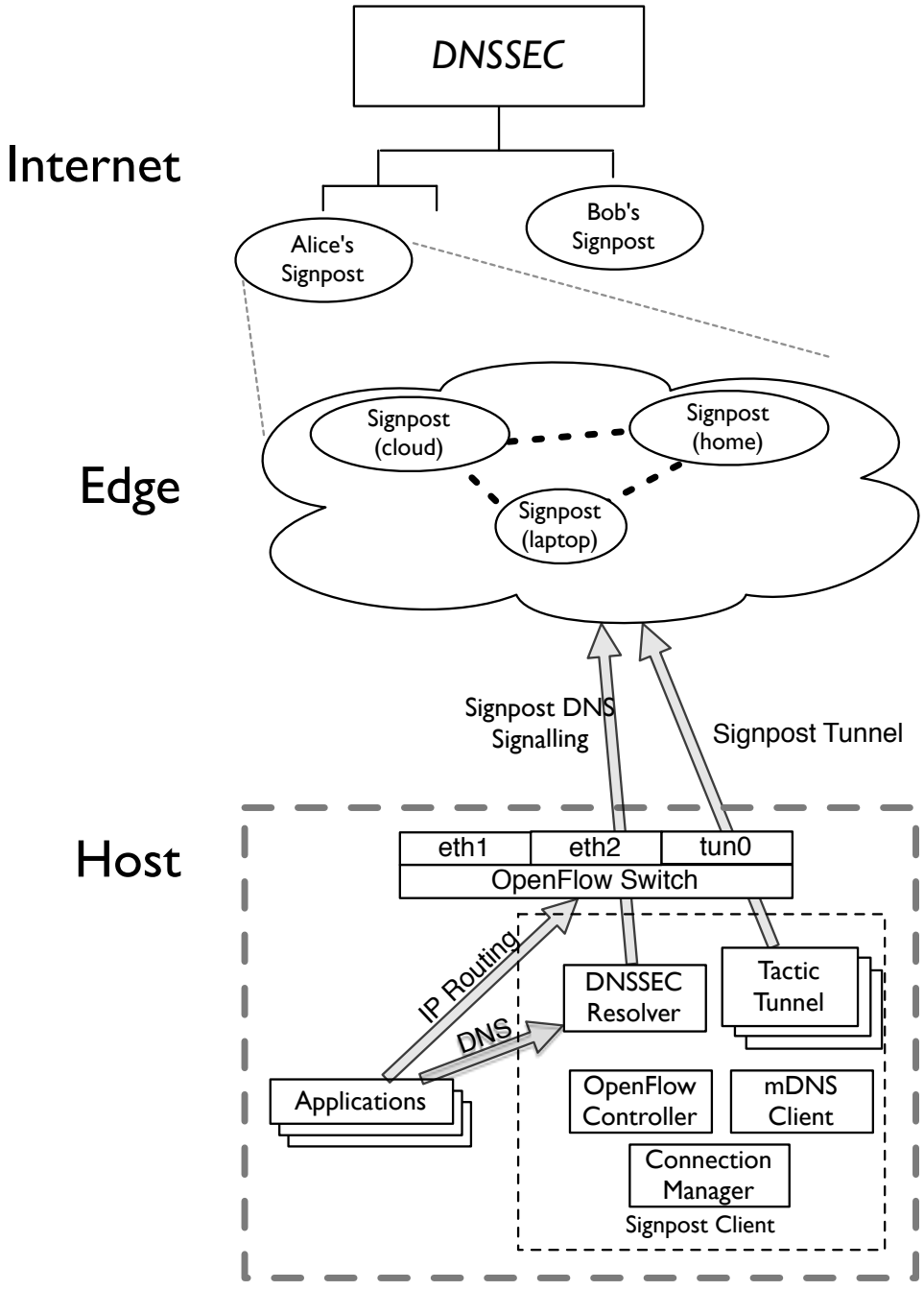


Figure 5.2: Signpost architecture

The system relies on an Internet-wide inter-device control channel, which enables connection capability and parameter negotiation. The architecture considers a *Controller Signpost* instance running on a well connected host, which bridges the device control channel across the Internet. Further, the system can establish connectivity between local devices without the mediation of the Signpost Controller. Signpost clients contain a Bonjour-based discovery mechanism, through which devices can establish ad-hoc secure and authenticated paths. In the upper layer of Figure 5.2, we present the naming organisation of the Signpost architecture. The system reuses the naming abstraction of the DNS service. Each device has a global domain name, while the domain hierarchy and name aliasing expresses the control relationship between devices and users. We extend the normal name resolution functionality of the DNS protocol and introduce the *Effectful name resolution* operation for Signpost-enable domains; a name resolution expresses the interest of a user to establish an end-to-end path (Subsection 5.3.4). For the rest of the section we present in depth the details of the Signpost system.

5.3.1 Network Tactic

Currently the network community offers a wide selection of software to establish connectivity between network devices. In Table 5.1, we present a small survey of such mechanisms along with their network requirements and security properties. From the data of the table, we note the high diversity between connection properties. For example, a significant subset of the mechanisms abuse application layer protocol functionality in order to establish IP connectivity, thus allowing users to bypass strict network policies. An equally significant subset of the mechanisms focus on encryption and privacy enhancement of end-to-end Internet paths, while a third class of these mechanisms enables connectivity through simple service advertisement. Further, the mechanisms vary significantly on the network layer they operate and the type of connection they provide. Tunnels provide connectivity on a specific port of the transport layer, or introduce an overlay network on the network layer. The majority of the tunelling mechanism functions over UDP or TCP protocol, while there are protocols that function over lower layer protocols, like ICMP. Finally, authentication of users is not uniform. Approaches vary from user-based authentication, using either passwords or certificates, to simple passphrase checks, while a subset of the protocols doesn't

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

Table 5.1: Tactics table.

support any authentication.

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

Each tactic in Signpost is modelled as a 4 state automaton. The state space diagram is presented in Figure 5.3. A tactic is initialized in the Idle state. A test method invocation transfers the tactic to the testing state, while executing the testing logic of the tactic. The test method performs tactic specific testing in order to detect the limitations in network connectivity and the configuration required in order to establish connectivity. If testing is successful, the tactic can progress to the connect state which will configure an end-to-end path. Once the end-to-end path is setted up, then the Tactic can progress to the enabled state and forward packets to the end-to-end path.

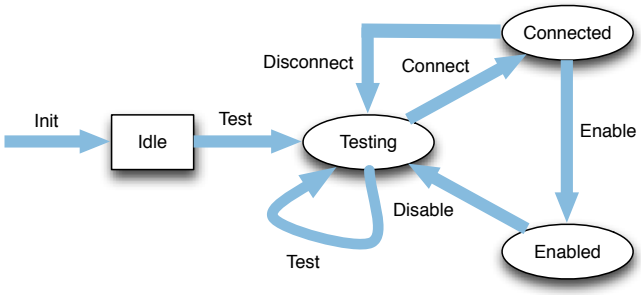


Figure 5.3: Signpost tactic lifecycle

Finally, the tactic automata provides methods to backtrack from each state and clear stored state. In order to avoid packet reordering, Signpost permits parallel existence of multiple connected tactics for a set of devices, but a single tactic can be enabled at any point. *Mention that tactic is able to control OpenFlow also*

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

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

Discuss weight of tactics

5.3.2 Forwarding

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

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

5.3.3 Connection Manager

Signpost is able to establish multiple end-to-end paths between devices through tactic synthesis. Unfortunately, multipath connectivity is not supported by popular transport protocols and newer protocols like SCTP and multipath TCP, that support multipath connectivity, are not available in production applications. Multipath functionality in Signpost could be integrated in existing transport protocol through careful OpenFlow flow manipulation, but because performance is not homogenous between paths, packet

reordering may occur and reduce network stack functionality. As a result, Signpost we establish and use a single end-to-end path between any two devices. Path search and establishment is encapsulated in the Connection Manager Subsystem.

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

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

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

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

5.3.4 Effectful Naming

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

Domain naming follows a simple organisation model which though provides very good scaling properties. Every domain name consist of a sequence of name token which are organised in a hierarchical tree structure with a single root, the empty string. Every node on the tree can be coupled with a number of DNS Resource Records (RR) which define information for the specific domain name like network addresses, naming aliases, service information and many more. In order to scale the naming service efficiently across the Internet, the protocol provides a specific RR type, the NS record, which delegates control for a domain to a specific set of DNS servers. Querying the domain tree for a specific RR requires at least the address of a DNS server. The DNS client can then follow service redirections in order to find a server responsible for the

requested domain. Additional, the explicit caching mechanism of the service reduces significantly the load on naming servers.

Secure Names For All Internet Users In the initial definition of the naming service, the protocol provided very weak security guarantees. In order to enhance the security primitives the IETF standardised a number of DNS protocol extensions, which ascribe as DNSSEC. DNSSEC defines several extra RR types Arends et al. [2005] used to provide a signing chain that can authenticate DNS records. In essence, a zone signs its authoritative RRsets using its private key, and then publishes its public key via a DNSKEY RR to allow resolvers to validate the RRset signatures. Following the signing of the root zone, and certain top level domains beneath it, a chain of trust is formed back to the root, whereby a given resolver can be certain that the response it has received to a query *is* the correct response from the authoritative DNS server for the name in question. The resolver requires, as in the X509 certificate architecture, only a list of authenticated anchors configured out-of-band of the protocol and injected in the authentication chain.

In terms of the Signpost system, we have been delegated control of the `.signpost.io` domain and registered a signing key with the `.io` domain. For each user of the Signpost system, we delegate and authenticate control to a subdomain and redirect DNS queries to the Signpost Controller of the user personal cloud, where the user can register its devices. As an example with reference to Figure 5.2, user Alice is granted control of the domain `alice.signpost.io` where she can register her laptop under the domain name `laptop.alice.signpost.io`. By running a Signpost server, an individual has a globally accessible authenticated public identity on the Internet via their public-private key-pair. Using this key-pair, a user can sign and authenticate messages, bootstrap public key cryptography mechanisms and run key-exchange mechanisms such as Diffie-Merkle-Hellman Rescorla [1999] and derive new shared private keys between any two devices over unsecure channels.

The base DNSSEC RR types, defined in Arends et al. [2005], provide a mechanism to authenticate the channel from the server to the client. In terms of Signpost, we are also interested to enable an authenticated channel from the client to the server. This will permit to the server to present a different view over the resource mappings,

depending on the querying entity. [1] Signpost uses the SIG(0) RR type, defined in Eastlake [2000], which functions as a signature on a request. The record was introduced in order to allow authorized clients to update RR records on an authoritative server, while it permits a client also to point to the signing entity, in order to fit authentication with the DNSSEC key structure.

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

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

discnnected Signpost functionality

Using the DNS protocol we are also able to ensure the functionality of the system when devices are disconnected from the Signpost Controller, while connected in the same network. Signpost uses DNS-based service discovery (DNS-SD) Cheshire and Krochmal [2011b] in order to enable device local connectivity. DNS-SD is a specification of DNS RR organisation which enables efficient service advertisement and browsing and along with Multicast-DNS Cheshire and Krochmal [2011a] they establish an efficient local service discovery mechanism. The combination of DNS-SD and

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

multicast-DNS is currently a widely used mechanism for service discovery in local networks and supported by most available operating systems. Signpost registers and advertises a service record for the Signpost service with name _sp._tcp.local in the local network with a target the domain name of the device, along with an RRSIG RR and the DNSKEY RR of the device, signed with the private key of the user. Using these records a Signpost client is able to verify a destination Signpost service and establish a control channel. The device will use the service advertisement information when a name lookup is performed. During a name lookup, the destination device will function as a Signpost Controller responsible for the specific device only and employ all the logic described previously.

5.3.5 Security and Key Management

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

5.4 Evaluation

In this section we analyse the performance of the Signpost system and its impact in the functionality of traditional Personal cloud applications. We, present the implementation details of the system (Subsection 5.4.1), measure the performance of the Signpost tactics over the Internet (Subsection 5.4.2) and present a small scale study of the functionality of current application over the Signpost system.

5.4.1 Signpost implementation

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

Signpost provides support for the following tactics:

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

- *OpenVpn*: The tactic enables communication paths using the OpenVpn tunneling mechanism. During connection, Signpost devices will use the Signpost key infrastructure to generate public keys and bootstrap the OpenVPN authentication mechanism. Because of some limitation on the certificate chain evaluation of OpenSSL, during a connection the tactic will generate transient private keys which will be signed by the user private key, while the tactic will inject in the trusted certificate keychain a key certificate of the destination device signed by

the private key of the device. The tactic configure the OpenVpn software to expose connectivity over an ethernet TUN/TAP device, which is added under the OpenFlow switch control and normal OpenFlow packet forwarding rule establish full bidirection connectivity. *Mention OpenVPN ARP cache*

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

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

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

will initiate a TCP connection with the local SOCKS proxy with a request to establish a path between the two devices. In parallel, once the TCP connection is established with the local SOCKS proxy, the tactic will respond to the initial SYN request with a SYNACK and advertise a zero window in order to suppress any further data transmission from the application. Once a SOCKS response is returned by the SOCKS proxy, the tactic will either respond to the initial TCP flow with an ACK with a non-zero window and insert appropriate OpenFlow rules to permit connectivity between the two device over the SOCKS tunnel, or the client will inject a TCP RST if the SOCKS response was unsuccessful.

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

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

Once the exact port-mapping is inferred on the controller, the information is propagate to both devices, which will insert appropriate OpenFlow commands to manipulate source and destination port renumbering and notify the end-points to resume data transmission using a ACK with a non-zero window size.

5.4.2 Tunnel Evaluation

5.4.3 Application compatibility

5.5 Conclusions

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

Chapter 6

My Conclusions ...

Here I put my conclusions ...

References

B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, H. Levkowetz, and Ed. Extensible Authentication Protocol (EAP). RFC 3748, IETF, June 2004. 77

Aditya Akella, Srinivasan Seshan, and Anees Shaikh. An empirical evaluation of wide-area internet bottlenecks. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, IMC '03, pages 101–114, New York, NY, USA, 2003. ACM. ISBN 1-58113-773-7. doi: 10.1145/948205.948219. URL http://doi.acm.org/10.1145/948205.948219. 84

D. S. Alexander, W. A. Arbaugh, M. W. Hicks, P. Kakkar, A. D. Keromytis, J. T. Moore, C. A. Gunter, S. M. Nettles, and J. M. Smith. The switchware active network architecture. *Netwrk. Mag. of Global Internetwkg.*, 12(3):29–36, May 1998a. ISSN 0890-8044. doi: 10.1109/65.690959. URL http://dx.doi.org/10.1109/65.690959. 19

D Scott Alexander, Bob Braden, Carl A Gunter, Alden W Jackson, Angelos D Keromytis, Gary J Minden, and David Wetherall. Active network encapsulation protocol (anep). *URL http://www. cis. upenn. edu/switchware/ANEP/docs/ANEP. txt*, 1997a. 18

D. Scott Alexander, Marianne Shaw, Scott M. Nettles, and Jonathan M. Smith. Active bridging. In *Proceedings of the ACM SIGCOMM '97 conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '97, pages 101–111, New York, NY, USA, 1997b. ACM. ISBN 0-89791-905-X. doi: 10.1145/263105.263149. URL http://doi.acm.org/10.1145/263105.263149. 19

D.S. Alexander, W.A. Arbaugh, A.D. Keromytis, and J.M. Smith. Safety and security of programmable network infrastructures. *Communications Magazine, IEEE*, 36 (10):84–92, 1998b. ISSN 0163-6804. doi: 10.1109/35.722141. 19

L Andrerson, I. Minei, and B. Thomas. Rfc 5036: Ldp specification. 2007. 15

A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15(2), June 2005. 70

Roy Arends, Rob Austein, Matt Larson, Dan Massey, and Scott Rose. Rfc 4034: Resource records for the dns security extensions. *Internet Engineering Task Force*, 2005. 108

Patrik Arlos and Markus Fiedler. A method to estimate the timestamp accuracy of measurement hardware and software tools. In *PAM*, 2007. 33

D Awduche, Lou Berger, D Gan, Tony Li, Vijay Srinivasan, and George Swallow. Rfc 3209: Extensions to rsvp for lsp tunnels. *Network Working Group*, 64, 2001. 15

Giacomo Balestra, Salvatore Luciano, Maurizio Pizzonia, and Stefano Vissicchio. Leveraging router programmability for traffic matrix computation. In *Proc. of PRESTO workshop*, 2010. 44

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

Richard Bellman. On a routing problem. Technical report, DTIC Document, 1956. 16

Gary Berger. NodeFlow: An OpenFlow Controller Node Style. http://garyberger.net/?p=537, 2012. 26

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

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

A. Bianco, R. Birke, L. Giraudo, and M. Palacin. Openflow switching: Data plane performance. In *IEEE ICC*, may 2010. 36

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

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

Stewart Brand. *How Buildings Learn: What Happens After They're Built*. Penguin, 1995. 64

British Telecoms. Broadband usage policy. http://bt.custhelp.com/app/answers/detail/a_id/10495/~/broadband-usage-policy, 2013. 85

Broadcom. Bradcom switching silicon chips, 2013. URL http://www.broadcom.com/products/Switching/Data-Center. 12

Martin A. Brown. Pakistan hijacks YouTube. http://www.renesys.com/blog/2008/02/pakistan_hijacks_youtube_1.shtml, 2008. 17

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. 66

Z Cai, AL Cox, and TS Eugene Ng. Maestro: balancing fairness, latency and throughput in the openflow control plane. Technical report, Rice University Technical Report TR11-07, 2011. 55

Brian Carpenter and Scott Brim. Rfc 3234: Middleboxes: Taxonomy and issues. 2002. 95, 109

Cbench. Cbench: controller benchmark, 2010. URL http://docs.projectfloodlight.org/display/floodlightcontroller/Cbench. 47, 55

Y. Chae and E. Zegura. Canes: An execution environment for composable services. In *Proceedings of the 2002 DARPA Active Networks Conference and Exposition*, DANCE '02, pages 255–, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7695-1564-9. URL http://dl.acm.org/citation.cfm?id=874028.874134. 19

Stuart Cheshire and Marc Krochmal. Rfc 6762: Multicast dns. *Work in Progress*, 2011a. 109

Stuart Cheshire and Marc Krochmal. Rfc 6763: Dns-based service discovery. 2011b. 109

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. 64

Marshini Chetty, Richard Banks, Richard Harper, Tim Regan, Abigail Sellen, Christos Gkantsidis, Thomas Karagiannis, and Peter Key. Who's hogging the bandwidth: the consequences of revealing the invisible in the home. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 659–668, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-929-9. doi: 10.1145/1753326.1753423. URL http://doi.acm.org/10.1145/1753326.1753423. 64, 66, 88

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. 63

Lucas Di Cioccio, Rennate Teixeira, and Catherine Rosenberg. Characterizing home networks with homenet profiler. Technical report, Technicolor, 2011. 61, 62

Cisco. Cisco express forwarding overview. http://www.cisco.com/en/US/docs/ios/12_2/switch/configuration/guide/xcfcef.html. 13

Cisco. Cisco asr 9000 series ethernet line cards. http://www.cisco.com/en/US/prod/collateral/routers/ps9853/data_sheet_c78-501338.html, 2012a. 87

Cisco. Per VLAN Spanning Tree (PVST). http://www.cisco.com/en/US/tech/tk389/tk621/tk846/tsd_technology_support_sub-protocol_home.html, 2012b. 15

Cisco. Per VLAN Spanning Tree Plus(PVST+). http://www.cisco.com/en/US/tech/tk389/tk621/tk847/tsd_technology_support_sub-protocol_home.html, 2012c. 15

I Cisco. Cisco Visual Networking Index: Forecast and Methodology, 2011–2016. *CISCO White paper*, 2012d. 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. 35

A. Crabtree, T. Rodden, T. Hemmings, and S. Benford. Finding a place for ubicomp in the home. In *Proc. UbiComp*, 2003. 65, 67

Simon Crosby, Sean Rooney, Rebecca Isaacs, and Herbert Bos. A perspective on how atm lost control. *SIGCOMM Comput. Commun. Rev.*, 32(5):25–28, November 2002. ISSN 0146-4833. doi: 10.1145/774749.774754. URL http://doi.acm.org/10.1145/774749.774754. 22

S. da Silva, Y. Yemini, and D. Florissi. The netscript active network system. *Selected Areas in Communications, IEEE Journal on*, 19(3):538–551, 2001. ISSN 0733-8716. doi: 10.1109/49.917713. 19

DARPA. Active Networks. URL http://www.sds.lcs.mit.edu/darpa-activenet/. 17

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

D.S. Decasper, B. Plattner, G.M. Parulkar, Sumi Choi, J.D. DeHart, and T. Wolf. A scalable high-performance active network node. *Network, IEEE*, 13(1):8–19, 1999. ISSN 0890-8044. doi: 10.1109/65.750445. 20

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

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

E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959. ISSN 0029-599X. doi: 10.1007/BF01386390. URL http://dx.doi.org/10.1007/BF01386390. 16

Marcel Dischinger, Andreas Haeberlen, Krishna P. Gummadi, and Stefan Saroiu. Characterizing residential broadband networks. *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement - IMC '07*, page 43, 2007a. doi: 10.1145/1298306.1298313. URL http://portal.acm.org/citation.cfm?doid=1298306.1298313. 62

Marcel Dischinger, Andreas Haeberlen, Krishna P Gummadi, and Stefan Saroiu. Characterizing residential broadband networks. In *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. ACM Request Permissions, October 2007b. 5, 84

R. Droms. Dynamic Host Configuration Protocol. RFC 2131, IETF, March 1997. 73

DSLreports. What is powerboost? http://www.dslreports.com/faq/14520. 63

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

Donald E Eastlake. Rfc 2931: Dns request and transaction signatures (sig (0) s). 2000. 109

Emulab. Emulab: Network emulation testbed, 2000. URL http://www.emulab.net. 48

Endace. Endace DAG Cards. http://www.endace.com/endace-dag-high-speed-packet-capture-cards.html. 35

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. 63

FCC. A Report on Consumer Wireline Broadband Performance in the U.S. http://www.fcc.gov/measuring-broadband-america/2012/july. 62

Open Network Foundations. https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf, 2012. 22

Jim Gettys and Kathleen Nichols. Bufferbloat: Dark buffers in the internet. *Queue*, 9(11):40:40–40:54, November 2011. ISSN 1542-7730. doi: 10.1145/2063166.2071893. URL http://doi.acm.org/10.1145/2063166.2071893. 47

Google. OpenFlow @ Google. http://www.opennetsummit.org/archives/apr12/hoelzle-tue-openflow.pdf, 2013. 23

R.E. Grinter and W.K. Edwards. The work to make the home network work. In *Proc. ECSCW*, 2005. 64

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. 34

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. 26, 55, 69

Robert W Hahn and Scott Wallsten. The economics of net neutrality. *The Economists' Voice*, 3(6), 2006. 83

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. 32, 46

J.J. Hartman, P.A. Bigot, P. Bridges, B. Montz, R. Piltz, O. Spatscheck, T.A. Proebsting, L.L. Peterson, and A. Bavier. Joust: a platform for liquid software. *Computer*, 32(4):50–56, 1999. ISSN 0018-9162. doi: 10.1109/2.755005. 19

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. 62

Thomas R. Henderson, Sumit Roy, Sally Floyd, and George F. Riley. ns-3 project goals. In *Proceeding from the 2006 workshop on ns-2: the IP network simulator*, WNS2 '06, New York, NY, USA, 2006. ACM. ISBN 1-59593-508-8. doi: 10.1145/1190455.1190468. URL http://doi.acm.org/10.1145/1190455.1190468. 49, 54

M. Hicks, J.T. Moore, D.S. Alexander, C.A. Gunter, and S.M. Nettles. Planet: an active internetwork. In *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1124–1133 vol.3, 1999. doi: 10.1109/INFCOM.1999.751668. 19

Michael Hicks, Pankaj Kakkar, Jonathan T. Moore, Carl A. Gunter, and Scott Nettles. Plan: a packet language for active networks. *SIGPLAN Not.*, 34(1):86–93, September 1998. ISSN 0362-1340. doi: 10.1145/291251.289431. URL http://doi.acm.org/10.1145/291251.289431. 18

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, 95

C. Hopps. Rfc 2992: Analysis of an equal-cost multi-path algorithm. 2000. 28

HP. Provision^{TM}asic: Built for the future. URL http://www.hp.com/rnd/itmgrnews/built_for_future.htm. 12

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

IEEE. IEEE Std 802.1D-1998: Media Access Control (MAC) Bridges. http://standards.ieee.org/getieee802/download/802.1D-1998.pdf, 1998. 14

IEEE. IEEE Std 802.1D-2004: Media Access Control (MAC) Bridges. http://standards.ieee.org/getieee802/download/802.1D-2004.pdf, 2004. 14

IEEE. IEEE Std 802.1Q: Virtual Bridged Local Area Networks. http://www.dcs.gla.ac.uk/~lewis/teaching/802.1Q-2005.pdf, 2005. 15

IEEE. Ieee 802.1 aq shortest path bridging. 2006. 15

Intel. Intel ethernet switching components. URL http://ark.intel.com/products/family/134/Intel-Ethernet-Switching-Components. 12

Teerawat Issariyakul. *Introduction to network simulator NS2*. Springer Science+ Business Media, 2012. 48

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

M. Jarschel, F. Lehrieder, Z. Magyari, and R. Pries. A flexible openflow-controller benchmark. In *Software Defined Networking (EWSDN), 2012 European Workshop on*, pages 48–53, 2012. doi: 10.1109/EWSDN.2012.15. 34

Lavanya Jose, Minlan Yu, and Jennifer Rexford. Online measurement of large traffic aggregates on commodity switches. In *Proc. of the USENIX HotICE workshop*, 2011. 44

Juniper. T series core router architecture overview. http://www.slideshare.net/junipernetworks/t-series-core-router-architecture-review-whitepaper, 2012. 13

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

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

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

Masayoshi Kobayashi, Srini Seetharaman, Guru Parulkar, Guido Appenzeller, Joseph Little, Johan van Reijendam, Paul Weissmann, and Nick McKeown. Maturing of OpenFlow and Software Defined Networking through Deployments . URL http://yuba.stanford.edu/openflow/documents/openflow_deployment_journal_paper_aug2012.pdf. 23, 40

Christian Kreibich, Nicholas Weaver, Boris Nechaev, and Vern Paxson. Netalyzr: illuminating the edge network. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, IMC '10, pages 246–259, New York, NY, USA,

2010. ACM. ISBN 978-1-4503-0483-2. doi: 10.1145/1879141.1879173. URL http://doi.acm.org/10.1145/1879141.1879173. 63, 95

Balachander Krishnamurthy and CE Wills. On the leakage of personally identifiable information via online social networks. *...ACM workshop on Online social networks*, 2009. URL http://dl.acm.org/citation.cfm?id=1592668. 97

Nate Kushman, Srikanth Kandula, and Dina Katabi. Can you hear me now?!: it must be bgp. *SIGCOMM Comput. Commun. Rev.*, 37(2):75–84, March 2007. ISSN 0146-4833. doi: 10.1145/1232919.1232927. URL http://doi.acm.org/10.1145/1232919.1232927. 17

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

Anna Leach. Greedy Sky admits: We crippled broadband with TOO MANY users, 2013. 85

John T. Lewis, Raymond Russell, Fergal Toomey, Brian McGurk, Simon Crosby, and Ian Leslie. Practical connection admission control for ATM networks based on on-line measurements. *Computer Communications*, 21 (17):1585 – 1596, 1998. ISSN 0140-3664. doi: 10.1016/S0140-3664(98)00224-2. URL http://www.sciencedirect.com/science/article/pii/S0140366498002242. 22

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 2

Lwt. cooperative threads library for ocaml, 2013. URL http://ocsigen.org/lwt/. 50

Anil Madhavapeddy, Richard Mortier, Charalampos Rotsos, David Scott, Balraj Singh, Thomas Gazagnaire, Steven Smith, Steven Hand, and Jon Crowcroft. Unikernels:

Library operating systems for the cloud. In *Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems*, pages 461–472. ACM, 2013. 31, 55

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. 63

G. Malkin. RFC 2453: RIP version 2, November 1998. 16

ML Mazurek, JP Arsenault, and Joanna Bresee. Access control for home data sharing: Attitudes, needs and practices. *Proceedings of the 28th* ..., pages 645–654, 2010. URL http://dl.acm.org/citation.cfm?id=1753421. vii, 65

S Merugu, S Bhattacharjee, Y Chae, M Sanders, K Calvert, and E Zegura. Bowman and canes: Implementation of an active network. In *PROCEEDINGS OF THE ANNUAL ALLERTON CONFERENCE ON COMMUNICATION CONTROL AND COMPUTING*, volume 37, pages 147–156. Citeseer, 1999. 19

Microsoft. Windows filtering platform, 2012. URL http://msdn.microsoft.com/en-us/library/windows/desktop/aa366510.aspx. 86

D L Mills and H W Braun. The NSFNET backbone network. *ACM SIGCOMM Computer Communication Review*, 17(5):191–196, 1987. 2

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

P. V. Mockapetris. RFC 1034: Domain names — concepts and facilities, November 1987. 107

A.B. Montz, D. Mosberger, S.W. O'Mally, L.L. Peterson, and T.A. Proebsting. Scout: a communications-oriented operating system. In *Hot Topics in Operating Systems, 1995. (HotOS-V), Proceedings., Fifth Workshop on*, pages 58–61, 1995. doi: 10.1109/HOTOS.1995.513455. 19

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. 69

J. Moy. RFC 2328: OSPF version 2, April 1998. 16

Jad Naous, David Erickson, G. Adam Covington, Guido Appenzeller, and Nick McKeown. Implementing an openflow switch on the netfpga platform. In *ANCS*, 2008. 37

The netfilter.org project. libnetfilter conntrack project, 2010. URL http://www.netfilter.org/projects/libnetfilter_conntrack/index.html. 86

Carrier Ethernet News. Upgrading to bigger backhaul bandwidth for dslams. http://www.carrierethernetnews.com/articles/319137/upgrading-to-bigger-backhaul-bandwidth-for-dslams/, 2011. 83

ofcom. The Consumer Experience of 2012, 2012. URL http://stakeholders.ofcom.org.uk/binaries/research/consumer-experience/tce-12/Consumer_Experience_Researc1.pdf. 62

OFLOPS. OFLOPS manual, 2011. http://www.openflow.org/wk/index.php/Oflops. 34

R. Olsson. *pktgen*, the linux packet generator. In *Proc. Linux Symposium*, 2005a. 78

R. Olsson. pktgen the linux packet generator. In *Proceedings of Linux symposium*, 2005b. 35

Sean W. O'Malley and Larry L. Peterson. A dynamic network architecture. *ACM Trans. Comput. Syst.*, 10(2):110–143, May 1992. ISSN 0734-2071. doi: 10.1145/128899.128901. URL http://doi.acm.org/10.1145/128899.128901. 17

OpenFlow Consortium. OpenFlow reference implementation. http://www.openflow.org/wp/develop/, 2009a. 37

OpenFlow Consortium. Openflow switch specification (version 1.0.0). www.openflow.org/documents/openflow-spec-v1.0.0.pdf, December 2009b. 36, 38, 41

D. Oran. RFC 1142: OSI IS-IS intra-domain routing protocol, February 1990. 16

Abhinav Pathak, Ming Zhang, Y. Charlie Hu, Ratul Mahajan, and Dave Maltz. Latency inflation with mpls-based traffic engineering. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, IMC '11, pages 463–472, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-1013-0. doi: 10.1145/2068816.2068859. URL http://doi.acm.org/10.1145/2068816.2068859. 15

Joshua Pelkey and George Riley. *Distributed Simulation with MPI in ns-3*. March 2011. 55

Radia Perlman. An algorithm for distributed computation of a spanningtree in an extended lan. *SIGCOMM Comput. Commun. Rev.*, 15(4):44–53, September 1985. ISSN 0146-4833. doi: 10.1145/318951.319004. URL http://doi.acm.org/10.1145/318951.319004. 14

Justin Pettit, Jesse Gross, Ben Pfaff, Martin Casado, and Simon Crosby. Virtualizing the network forwarding plane. In *DC-CAVES*, 2010. 36, 37, 57, 69

PlanetLab. PlanetLab: An open platform for developing, deploying and accessing planetary-scale services, 2007. URL http://www.planet-lab.org. 48

Ian Pratt. Pratt's Test TCP. http://www.cl.cam.ac.uk/research/srg/netos/netx/index.html, 2002. 52

Broadband Canada Program. Broadband canada: Connecting rural canadians - frequently asked questions. http://www.ic.gc.ca/eic/site/719.nsf/eng/h_00004.html, 2013. 85

DCAN project. DCAN - "Devolved Control of ATM Networks". http://www.cl.cam.ac.uk/research/srg/netos/old-projects/dcan/, 2000. 21

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. 63

Y. Rekhter. RFC 1265: BGP protocol analysis, October 1991. 16

E Rescorla. Rfc 2631: Diffie-hellman key agreement method. *Internet Engineering Task Force*, 1999. 108

Return Infinity. Baremetalos. http://www.returninfinity.com/baremetal.html, 2013. 51

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. 65

Tom Rodden and Steve Benford. The evolution of buildings and implications for the design of ubiquitous domestic environments. *Proceedings of the conference on Human factors in computing systems - CHI '03*, (5):9, 2003. doi: 10.1145/642614.642615. URL http://portal.acm.org/citation.cfm?doid=642611.642615. 64

S. Rooney. The hollowman: an innovative atm control architecture. In *Proceedings of the fifth IFIP/IEEE international symposium on Integrated network management V : integrated management in a virtual world: integrated management in a virtual world*, pages 369–380, London, UK, UK, 1997. Chapman & Hall, Ltd. ISBN 0-412-80960-5. URL http://dl.acm.org/citation.cfm?id=280039.280084. 21

S. Rooney, J.E. van der Merwe, S.A. Crosby, and I.M. Leslie. The tempest: a framework for safe, resource assured, programmable networks. *Communications Magazine, IEEE*, 36(10):42–53, 1998. ISSN 0163-6804. doi: 10.1109/35.722136. 21

E. Rosen, A. Viswanathan, and R. Callon. Rfc 3031: Multiprotocol label switching architecture. *MPLS Working Group*, 64, 2001. 15

Charles L. Rutgers. An introduction to IGRP. Technical report, The State University of New Jersey, Center for Computers and Information Services, Laboratory for Computer Science Research,, August 1991. 16

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

Beverly Schwartz, Alden W. Jackson, W. Timothy Strayer, Wenyi Zhou, R. Dennis Rockwell, and Craig Partridge. Smart packets: applying active networks to network management. *ACM Trans. Comput. Syst.*, 18(1):67–88, February 2000. ISSN 0734-2071. doi: 10.1145/332799.332893. URL http://doi.acm.org/10.1145/332799.332893. 18

Aman Shaikh and Albert Greenberg. Experience in black-box ospf measurement. In *ACM IMC*, 2001. 41

E. Shehan and W.K. Edwards. Home networking and HCI: What hath God wrought? In *Proc. ACM CHI*, 2007. 65

E. Shehan-Poole, M. Chetty, W.K. Edwards, and R.E. Grinter. Designing interactive home network maintenance tools. In *Proc. ACM DIS*, 2008. 64, 65

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. 32, 87

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. 63

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. 69, 70

Big Switch. Floodlight OpenFlow Controller. http://www.projectfloodlight.org/floodlight/, 2013. 26, 34

B. Teitelbaum and S. Shalunov. Why Premium IP Service Has Not Deployed (and Probably Never Will) . Technical report, Internet2, May 2002. URL http://qos.internet2.edu/wg/documents-informational/20020503-premium-problems-non-architectural.html. 7

Ajay Tirumala, Feng Qin, Jon Dugan, Jim Ferguson, and Kevin Gibbs. Iperf: The tcp/udp bandwidth measurement tool. 2005. 90

P. Tolmie, A. Crabtree, T. Rodden, C. Greenhalgh, and S. Benford. Making the home network at home: digital housekeeping. In *Proceedings ECSCW*, 2007. 64, 66, 67

Amin Tootoonchian, Monia Ghobadi, and Yashar Ganjali. OpenTM: traffic matrix estimator for openflow networks. In *PAM*, 2010. 44

Jacobus Erasmus van der Merwe. Open service support for ATM. Technical Report UCAM-CL-TR-450, University of Cambridge, Computer Laboratory, 15 JJ Thomson Avenue, Cambridge CB3 0FD, United Kingdom, phone +44 1223 763500, November 1998. iv, 20

András Varga and Rudolf Hornig. An overview of the omnet++ simulation environment. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, Simutools '08, pages 60:1–60:10, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). ISBN 978-963-9799-20-2. URL http://dl.acm.org/citation.cfm?id=1416222.1416290. 48

Virgin Media. Virgin media cable traffic management policy. http://help.virginmedia.com/system/selfservice.controller?CMD=VIEW_ARTICLE&ARTICLE_ID=2781&CURRENT_CMD=SEARCH&CONFIGURATION=1002&PARTITION_ID=1&USERTYPE=1&LANGUAGE=en&COUNTY=us&VM_CUSTOMER_TYPE=Cable. 85

David Watson, Farnam Jahanian, and Craig Labovitz. Experiences with monitoring ospf on a regional service provider network. In *Proceedings of the 23rd International Conference on Distributed Computing Systems*, ICDCS '03, pages 204–,

Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1920-2. URL http://dl.acm.org/citation.cfm?id=850929.851939. 17

Paul Weissmann and Srini Seetharaman. How mature is OpenFlow to be introduced in production networks, 2011. URL http://changeofelia.info.ucl.ac.be/pmwiki/uploads/SummerSchool/Program/session_004.pdf. 32

D.J. Wetherall, John V. Guttag, and D.L. Tennenhouse. Ants: a toolkit for building and dynamically deploying network protocols. In *Open Architectures and Network Programming, 1998 IEEE*, pages 117–129, 1998. 18

Mike P. Wittie, Veljko Pejovic, Lara Deek, Kevin C. Almeroth, and Ben Y. Zhao. Exploiting locality of interest in online social networks. In *Proceedings of the 6th International COnference*, Co-NEXT '10, pages 25:1–25:12, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0448-1. doi: 10.1145/1921168.1921201. URL http://doi.acm.org/10.1145/1921168.1921201. 98

Xen Project. XAPI. http://www.xenproject.org/developers/teams/xapi.html. 53

Kok-Kiong Yap, Masayoshi Kobayashi, David Underhill, Srinivasan Seetharaman, Peyman Kazemian, and Nick McKeown. The stanford openroads deployment. In *Proceedings of ACM WINTECH*, 2009. 46

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. 62

Minlan Yu, Jennifer Rexford, Michael J. Freedman, and Jia Wang. Scalable flow-based networking with difane. In *ACM SIGCOMM*, August 2010. 32