Initial Thoughts on the Waypoint Service

Glen Gibb
grg@stanford.edu

Hongyi Zeng
hyzeng@stanford.edu

Nick McKeown
nickm@stanford.edu

Stanford University
Stanford, CA 94305

ABSTRACT

Network operators want additional functionality from the networks they manage. The current approach to add functionality is to deploy middleboxes. Unfortunately middleboxes raise concerns regarding robustness, correctness, and efficiency due to their need to be deployed at chokepoints.

This paper provides some initial thoughts of solving the middlebox problem in an architectural way. We believe that waypoint services are the correct way to add functionality to a network. Network processing can be modeled as classification followed by action. Additional functionality should be added to the network through a service model exposed as new actions. Services would be implemented at waypoints which reside off the normal packet path; routers can send traffic to those services for additional processing.

The waypoint service model allows services to be hosted anywhere within the network, allows services to be shared by multiple routers, and is accessible via a simple action API. Abstracting custom packet processing as waypoint services provides a systematic way to bring new functionalities to the networks.

1. INTRODUCTION

Middleboxes are devices that provide functionality beyond basic forwarding to a network. Middleboxes are used to incorporate features such as intrusion detection or address translation within the network. The name middlebox derives from the requirement to place the device in the network, specifically on a chokepoint through which all traffic of interest flows (Figure 1). Placing middleboxes along chokepoints raises several problems: how do you identify appropriate locations through which all traffic flows, and how do you maintain network connectivity if a middlebox fails? Despite these problems, middlebox deployment is currently the only viable method for adding additional features to a network.

Numerous proposals exist to overcome the shortcomings of chokepoint deployments [9, 15, 26]. The common idea amongst these is the waypoint: middleboxes are deployed *off* the normal traffic path, with traffic being explicitly routed through the waypoint before delivery to the destination (Figure 2). Waypoints eliminate the need to identify an appropriate chokepoint and allow the network to continue functioning after waypoint failure.

What is the right partition of functionality between waypoints and the network itself? Routers have clearly been growing in functionality over the years, unfortunately making them bloated in the process. At the other extreme, overlays have been proposed to use the regular internet merely to bring traffic to complex processing points. But this doesn't give much control over the network itself—you have to rely on the placement of the overlay nodes and the routing imposed by the network to get there. Ideally we'd like some combination of these approaches: we'd like complex processing to be performed outside of routers so that routers remain lean and efficient, and we'd like to have control over how traffic flows between our processing nodes within the network.

Recently, several flow-based approaches have been sug-

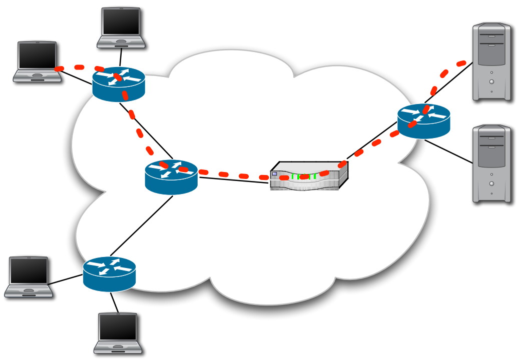


Figure 1: Middlebox deployment within a network. The middlebox must be deployed at a chokepoint within the network.

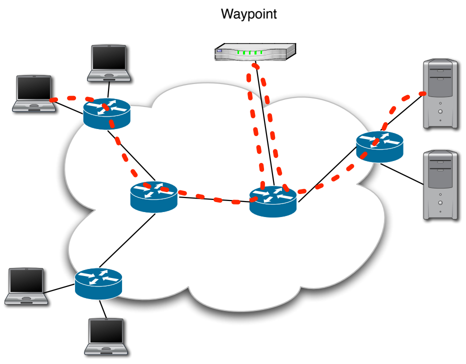


Figure 2: Waypoint processing within a network. Waypoint processors are deployed away from chokepoints; traffic is explicitly routed through waypoints before delivery to their final destination.

gested (e.g. OpenFlow [18], 4D [10]) in which the network datapath is kept very simple. Network primitives in every switch allow flows to be steered through the network by the network control plane. The set of primitives is very small, providing basic operations like `forward` to a port (or ports), `drop`, and `add/delete/modify` a field in a packet header. In the OpenFlow model, it is assumed that any complex processing is done in waypoints that may be deployed anywhere in the network. Waypoints can do the things current middleboxes do, such as NAT, firewalls, and DPI. Our goal is more than that: we want waypoints to be extensible so new services can easily be added by an experimenter, or by the owner/operator of a network to enhance service.

In this paper, we make an explicit case for moving the processing out of the datapath to waypoints, in an extensible manner that supports new services. We explain why this is a good idea, what such a waypoint system should look like, and how such a system can be controlled.

2. BACKGROUND

2.1 Middleboxes and waypoints

Network operators today deploy functionality within the network beyond standard IP routing via the use of middleboxes [5]. Middleboxes are generally built for a single purpose, for example, an application firewall or a deep packet inspection device. Middleboxes, as the name implies, are devices that are deployed within the network. They must be placed at a "chokepoint" within the network; this is a location between two routers through which all traffic to be processed flows (see Figure 1). Placing middleboxes at chokepoints raises several concerns:

Correctness Administrators must configure routing entries within the network to guarantee that the traffic of interest traverses the middlebox. This problem is compounded when several middleboxes exist within the network; ensuring the correct ordering and enforcement of policy is difficult with multiple middleboxes.

Efficiency The placement of middleboxes often forces unwanted traffic to traverse the middleboxes. The middleboxes should be configured to transparently pass through traffic not of interest, increasing the load on the middlebox and the delay through the network. The problem becomes particularly serious when deploying a middlebox on a network backbone.

Robustness A middlebox residing on the normal data path introduces a single point of failure. A middlebox that fails due to hardware failure, misconfiguration, or power failure can cause catastrophic network failure.

The waypoint model (Figure 2) is an improved model for deploying processing elements[1]. Instead of deploying processing elements at chokepoints, they are deployed off the main traffic path. Processing elements can either hang off a router along the path, or they can be deployed remotely at any place within the network, provided that any requirements for latency through the network are met.

[1]We use the term "processing elements" (PEs) to represent middleboxes and other custom packet processing devices in general, such as PCs, NPUs or programmable hardware. We give the formal definition of PE in Section 3.3.1

Routers make decisions about where to send packets by comparing the packet headers and with the entries stored within the router's flow tables. Traffic can be redirected through processing elements for additional processing by installing the correct entries in the flow tables. Packets are forwarded along their normal physical path following processing and eventually reach their destinations.

The waypoint model solves the problems with middleboxes raised above. Robustness is improved as processing elements are aggregated behind a router, enabling the use of fail-over modules and fail-over routing rules to ensure continued network connectivity in the event of device failure. Efficiency and correctness are both addressed because policy installed within the routers determines alone what additional processing to apply to packets; contrast this with the middlebox approach where physical location combined with ad-hoc routing rules determine the processing. Only the packets requiring additional processing are sent to processing elements, and it is easy to determine which order processing elements will be traversed.

2.2 Related Work

The waypoint model can be classified as indirection in the networking data plane. Previous studies have shown that indirection systems can solve different networking problems.

The Internet Indirection Infrastructure (*i3*) [25] uses indirection to add services to the network, including the ability to perform custom processing along the path from source to destination. SelNet [8] is a network architecture that uses a virtualized link layer to support explicit indirection; particular elements within the network are able to modify packets that transit them.

Numerous researchers have suggested using the waypoint model for middlebox deployment. The Delegation-Oriented Architecture [26] proposes using delegation as a primitive to realize waypoint indirection and eventually avoid the harmful side-effects of middleboxes. PLayer [15] is a policy-aware Layer 2.5 network that enable one-hop waypoints designed for data centers. Similarly, Open Service Framework [9] enables an Ethernet switch to steer packets through middleboxes based on user specified rules that consist of fields from the packet header and the ingress port. Cisco proposes Unified Network Services/Service Insertion Architecture [7] to apply services to particular traffic.

Ethane [6] and OpenFlow [18] are flow-based architectures that both advocate using waypoints to perform complex processing. Flowstream [11] proposes the use of virtual machines and OpenFlow for parallel (load balancing) and serial (pipelining) packet processing. The In-Network Processing framework [16] takes advantage of the programmability of OpenFlow and uses virtual machines to replace middleboxes using the waypoint model.

3. WAYPOINT SERVICES

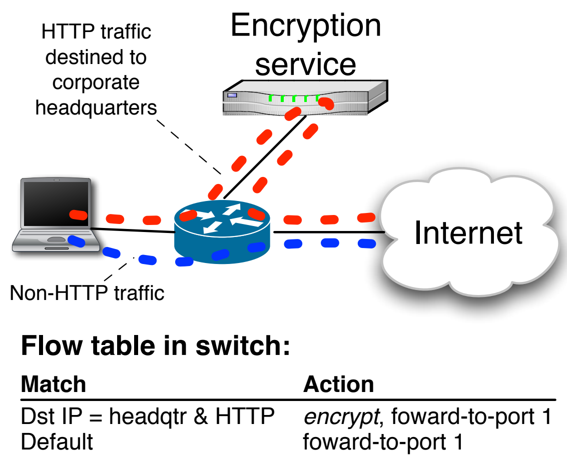
This section introduces the idea of a service model as a method of offering additional functionality within the network. We explain the rationale behind the service model, it's operation, and the feasibility of implementation today.

3.1 The service model

Functionality is added to the network today via middleboxes. The routers that compose the network are unaware of the middleboxes; middleboxes are placed in the physical path between routers and “hijack” traffic for additional processing. We posit that this is the wrong approach. Placing middleboxes on chokepoints introduces fragility into the network. Better decisions about when and how to use additional functionality can be made if the network is aware of that functionality.

We propose a service model for introducing functionality into the network. New functions are deployed as services which the network is aware of. Network administrators can “subscribe” to the service by sending traffic of one or more routers to that service for additional processing. Services themselves are exposed as additional actions that routers can perform on packets.

A simple application helps illustrate the model. A network administrator wishes to encrypt a subset of traffic before it enters the public internet. In the service model, the network administrator deploys an encryption service within the network (or requests a service from an external provider). The network is then configured to send all relevant traffic through the encryption service before transmission over the internet. Figure 3 and Figure 4 show the encryption service deployed within different topologies.



Match	Action
Dst IP = headqtr & HTTP	*encrypt*, foward-to-port 1
Default	foward-to-port 1

Figure 3: Deployment of a cryptographic service within a simple topology. Only HTTP traffic destined to the corporate headquarters is encrypted; all other traffic remains unencrypted.

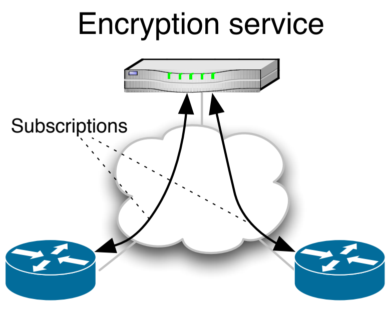


Figure 4: Deployment of a shared cryptographic service. Multiple switches route traffic to the encryption service for processing.

Services themselves are deployed as *waypoint processors* or simply *waypoints*. A waypoint processor is a device that attaches to the network that performs the processing necessary to implement the service. The encryption service waypoint from the example could conceptually be implemented as a process running on a commodity PC that encrypts all data.

Several fundamental questions arise regarding the service model. Where should services be deployed? How is traffic routed through a service? We provide answers to these questions later in this section.

3.2 Requirements of the model

We have identified four key requirements that the service model must meet:

Location agnostic Services should be locatable *anywhere* within the network. Network operators should not be constrained in where they deploy a service.

Simple API The service API must be intuitive and easy-to-use.

Sharable Services should be sharable amongst many subscribers. Operators may not wish or be able to provide dedicated resources for each router desiring a service.

Unmodified endhosts Operators wish to deploy new functionality without modifying endhosts. Operators may not even be able to modify the endhosts.

3.3 Service components

The waypoint service model consists of three main components: processing elements, a programmable interconnect, and a controller.

3.3.1 Processing elements

The processing element (PE) is the component that performs custom packet processing. A PE may be fixed-function or programmable. Existing middleboxes, such as NATs and firewalls, can be incorporated as fixed-function devices, as too can new custom-designed fixed-function components. Programmable elements that can be incorporated include CPUs, NPUs, and FPGAs; these devices allow truly customizable processing and enable the available functionality to change in response to the system’s needs.

PEs are free to implement their specified functionality in any manner. The implementation of a PE will affect its performance which may have consequences for the performance of the network as a whole. We deliberately avoid addressing a number of issues in this paper due to space constraints including performance and the details of using programmable elements.

PEs consist of both data-plane and control-plane components. The data-plane component is responsible for packet processing. The control-plane component interacts with the controller (Section 3.3.3) to allow configuration of the PE. In the case of programmable PEs, the control plane provides the ability to program the device (although this functionality may not be exposed directly to the network operator).

A waypoint may contain more than one PE. A waypoint that contains multiple PEs may offer multiple services, or may offer a single service, utilizing the multiple PEs for scalability or construction of complex services.

Services can be shared by multiple switches by routing traffic from multiple sources to a single waypoint. PEs are

Table 1: Example flow table from a user perspective

Input Port	Src IP	Dst IP	Protocol	Action
*	192.168.0.0/24	*	TCP	`IDS, NAT`
*	192.168.1.0/24	*	*	`HTTP Proxy`
*	*	192.168.2.0/24	UDP	`drop`
0	*	*	*	`forward` Port 1

agnostic to the origin of traffic—if traffic can be routed to a PE it will process it. Figure 4 shows the encryption service being shared by multiple switches.

3.3.2 Programmable interconnect

The network must be capable of routing traffic to services anywhere within the network without modification to any endhosts. Only the traffic of interest should be sent to services for processing; in the context of the encryption example, only HTTP traffic destined to the corporate headquarters should be processed by the encryption service.

A programmable interconnect provides the best mechanism for providing the necessary selective and controlled routing within the network[2]. A programmable interconnect is one in which the switches can be programmed *through some means* to send traffic matching a given pattern to the specified interface.

When a router wishes to use a particular service, routing entries are installed along the path between the switch and the waypoint to route traffic to and from the waypoint. The path between the router and the waypoint can be chosen to optimize for different metrics such as minimizing the hop count, minimizing latency, or maximizing bandwidth.

It should be noted that a waypoint with multiple PEs will contain a programmable interconnect element. An internal interconnect may be treated as an extension of the external interconnect, or it may be managed transparently to the operator, depending upon the mode of operation.

3.3.3 Controller

The programmable interconnect and the PEs are managed by an external controller.

The controller programs the interconnect to provide connectivity between switches and the services. The controller also programs the switches to provide connectivity between endpoints—we assume that the switches in the programmable interconnect are fully managed by the external controller rather than being controlled in part by local routing agents.

PE management performed by the controller consists of configuring PEs, and in the case of programmable PEs, programming them. The set of available configuration parameters varies depending upon the PE; in the encryption example, the encryption PE may allow the algorithm to be chosen and keys to be set.

The service model does not dictate the use of any particular controller or require any level of intelligence within the controller. The controller could be extremely simple, requiring all paths to be manually specified by the operator. Likewise the controller could be quite intelligent, automatically discovering the network topology and location of all services, and automatically installing routes according to policy without the intervention of an operator. Intelligent controllers offer considerable scope for further discussion but we defer that to a subsequent paper.

It should be noted that control may be delegated or partitioned between multiple controllers. As an example, a *waypoint service provider* may exist that offers waypoint services to customer networks. The waypoint service provide presumably has one or more large waypoints consisting of many PEs and a programmable interconnect. The waypoint service provider would operate their own controller, but would likely provide some level of visibility and delegation of control to the customer controller. Again due to space constraints we will discuss this in a later paper.

3.4 API

The API is a critical element of the service model: a network operator must be able to specify what services to apply to traffic and when. The API should be intuitive and easy-to-understand to enable the network operator to focus on the task of managing the network.

Network processing can be expressed as a combination of classification and action: for this set of traffic apply this set of actions. This is a very natural and intuitive way to think about processing in the network and we believe that this is the appropriate model for the API. Services should be exposed to the network as actions.

Considering just the programmable interconnect for a moment, the classification/action paradigm provides an appropriate API for configuration. Configuration occurs via the installation and removal of *flow table entries* from each switch. A flow table entry consists of a pattern (i.e. a set of header values) to match against incoming packets and a set of actions to apply to matching packets. Examples of actions available to the interconnect are `forward` and `drop`.

Services are accessed as additional actions available to each switch. The encryption service would provide an action `encrypt`; other examples could include `IDS` and `HTTP proxy`. Table 1 provides an example flow table that uses custom actions in some flows.

Rules with custom actions can't be installed directly in the physical switches, since the physical switches don't implement the custom actions. Flow table entries with custom actions must be translated into a series of flow table entries that route the traffic to and from the appropriate waypoint. This will result in the installation of flows in switches between the source switch and the waypoint when the waypoint is not directly connected to the switch requesting the service.

Translation between the desired flow tables and the physical flow tables can be performed automatically by an intelligent controller that knows the topology and location of services. Translation must be performed manually by the operator if they are using a non-intelligent controller.

[2]The service model can in theory be used with traditional IPv4 and other non-programmable networks by manipulating routing through active participation in the routing protocol message exchange. However, this approach is more complicated and error-prone than using a programmable interconnect.

4. FEASIBILITY

The waypoint service can be implemented using emerging technology today. The service model requires a programmable interconnect, a controller, and waypoints.

Programmable interconnects are known under the more common term today of software-defined network (SDN). OpenFlow [18] is an SDN technology that is currently available in commercial switches [14, 19]. OpenFlow controllers exist [2, 12] that can be easily adapted to support services and configure waypoints.

Waypoints can be built in several ways using CPUs, GPUs, NPUs, programmable hardware, and existing middleboxes. A simple waypoint may be constructed from an existing middlebox by connecting the middlebox to an OpenFlow switch. This allows off-path processing similar to the topology described in [15]. With this approach, network operators can reuse their existing resources by incorporating unmodified middleboxes into waypoints.

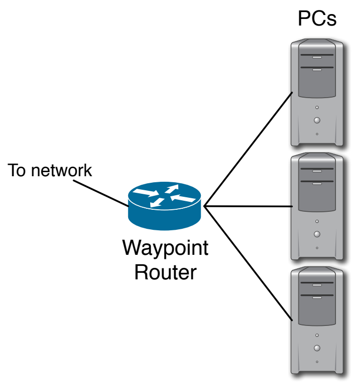


Figure 5: A waypoint processor containing an OpenFlow switch (Waypoint Router) and commodity PCs.

Waypoints can be built using commodity PCs as PEs (Figure 5). Any custom packet processing function can be implemented purely in software. Open Source implementations are available for many functions that may be desired from the network including firewalls [23], load balancers [13], proxies and caches [24], monitoring and measurement [20], intrusion detection [4, 21], and ubiquitous NAT. Concerns are often raised over the performance of software; the performance of a software service can be improved by utilizing a faster CPU (moving the service from a slow PE to a fast PE) or by running multiple instances of the software (utilizing multiple PEs). Spreading the processing load across multiple instances of the software requires the network to be capable of performing load balancing.

PC-based PEs can be used in a number of modes of operation. In the simplest mode, the entire PC is dedicated to a single service. In more complex modes, multiple services may be hosted on a single PC. Multiple services may be hosted in a single operating system using process isolation, or may be hosted in different virtual machines (Figure 6). Xen [1] and Open vSwitch [22] allow VMs be connected to a virtual OpenFlow switch, enabling VMs to be connected to the wider OpenFlow network. In addition to enabling multiple services to be hosted on a single PC, VMs also offer the ability to migrate services via VM migration, allowing services to be moved to lightly loaded machines in response

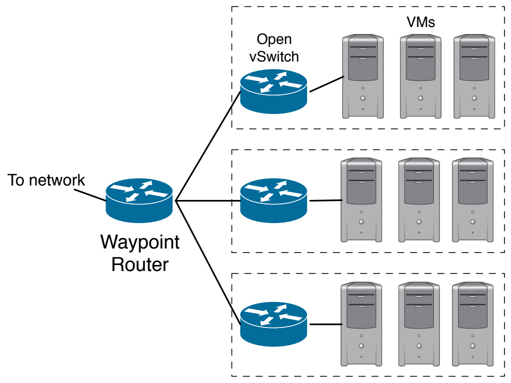


Figure 6: A waypoint processor with virtual machines, Open vSwitch (OVS) and Xen hypervisor.

to demand, and allowing services to be moved closer to the locations where they are used.

Programmable hardware devices such as the NetFPGA [17] are also ideal candidates for hosting services. Programmable hardware devices typically offer greater performance than is available in software, albeit at the cost of implementation complexity. Multiple services may be hosted on a single programmable hardware device when space permits; doing so requires either a dedicated port per service or a switch implemented inside the device to route traffic to the correct service.

Services can be hosted in other programmable devices, such as network processors (NPUs) and graphic coprocessors (GPUs). The primary requirement for hosting a service is the ability to communicate with the network.

Waypoints may consist of one or more processing elements and the processing elements need not be the same type (Figure 7). Network operators are free to choose whether they want to deploy a small waypoint at every node in the network, a small number of large waypoints shared throughout the network, or some hybrid of these.

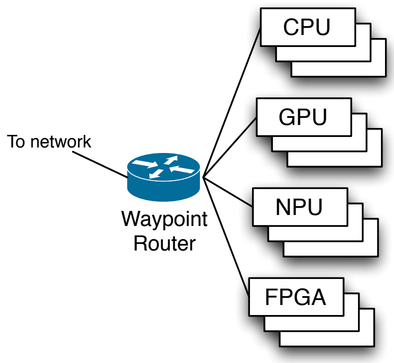


Figure 7: A waypoint processor with a mix of CPUs, GPUs, NPUs, and FPGAs.

5. CONCLUSION

Applications within the Internet, data center networks, and other computer networks are evolving rapidly. Many new applications require network functionality what was not forseen by designers [3]. Engineers have responded to this need by deploying middleboxes that implement the new functionality. We believe that ad-hoc middlebox deployment is problematic, and is neither sustainable nor economical in the long term. In this paper, we proposed the waypoint service, a new class of network service that routers can subscribe to that enables new functionality to be incorporated into the network. We discussed desired features of the service, the main building blocks, illustrated how today's network application can benefit from this service, and showed that the service model is feasible using current technologies.

We are currently implementing a prototype waypoint service system using OpenFlow switches, NetFPGAs, and commodity PCs.

6. REFERENCES

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 164–177, New York, NY, USA, 2003. ACM.

[2] Beacon: a Java-based OpenFlow Control Platform. `http://www.beaconcontroller.net/`.

[3] R. Braden and J. Postel. Requirements for Internet gateways. RFC 1009 (Historic). `http://www.ietf.org/rfc/rfc1009.txt`, June 1987. Obsoleted by RFC 1812.

[4] Bro Intrusion Detection System. `http://bro-ids.org/`.

[5] B. Carpenter and S. Brim. Middleboxes: Taxonomy and Issues. RFC 3234 (Informational). `http://www.ietf.org/rfc/rfc3234.txt`, Feb. 2002.

[6] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: taking control of the enterprise. *SIGCOMM Comput. Commun. Rev.*, 37:1–12, August 2007.

[7] Cisco Unified Network Services: Overcome Obstacles to Cloud-Ready Deployments. `http://www.cisco.com/en/US/solutions/collateral/ns340/ns517/ns224/ns836/ns976/white_paper_C11-64521.html`.

[8] R. Gold, P. Gunningberg, and C. Tschudin. A virtualized link layer with support for indirection. In *Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, FDNA '04, pages 28–34, New York, NY, USA, 2004. ACM.

[9] A. Gorti and V. Pandey. OSF - Open Service Framework. In *Proceedings of First Workshop on Data Center - Converged and Virtual Ethernet Switching*, DC-CAVES, 2009.

[10] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. A clean slate 4d approach to network control and management. *SIGCOMM Comput. Commun. Rev.*, 35:41–54, October 2005.

[11] A. Greenhalgh, F. Huici, M. Hoerdt, P. Papadimitriou, M. Handley, and L. Mathy. Flow processing and the rise of commodity network hardware. *SIGCOMM Comput. Commun. Rev.*, 39:20–26, March 2009.

[12] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: Towards and operating system for networks. In *ACM SIGCOMM Computer Communication Review*, July 2008.

[13] HAProxy - The Reliable, High Performance TCP/HTTP Load Balancer. `http://haproxy.1wt.eu/`.

[14] HP 5400zl/6600 series switches. `http://www.openflow.org/foswiki/bin/view/OpenFlow/Deployment/Vendor/HP`.

[15] D. A. Joseph, A. Tavakoli, and I. Stoica. A policy-aware switching layer for data centers. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, SIGCOMM '08, pages 51–62, New York, NY, USA, 2008. ACM.

[16] J. Lee, J. Tourrilhes, P. Sharma, and S. Banerjee. No more middlebox: integrate processing into network. *SIGCOMM Comput. Commun. Rev.*, 40:459–460, August 2010.

[17] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo. NetFPGA—an open platform for gigabit-rate network switching and routing. In *MSE '07: Proceedings of the 2007 IEEE International Conference on Microelectronic Systems Education*, pages 160–161, 2007.

[18] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38:69–74, March 2008.

[19] NEC IP8800/S3640 Ethernet switches. `http://www.necam.com/PFlow/`.

[20] ntop: Open Source Network Traffic Monitoring. `http://www.ntop.org/`.

[21] SNORT. `http://www.snort.org/`.

[22] Open vSwitch. `http://openvswitch.org/`.

[23] SmoothWall Express: Express Open Source Firewall Project. `http://www.smoothwall.org/`.

[24] squid : Optimising Web Delivery. `http://www.squid-cache.org/`.

[25] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet Indirection Infrastructure. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '02, pages 73–86, New York, NY, USA, 2002. ACM.

[26] M. Walfish, J. Stribling, M. Krohn, H. Balakrishnan, R. Morris, and S. Shenker. Middleboxes no longer considered harmful. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, pages 15–15, Berkeley, CA, USA, 2004. USENIX Association.