Flow processing and the rise of commodity network hardware

Adam Greenhalgh[‡] Mark Handley[‡] Mickael Hoerdt[§]
Felipe Huici[*] Laurent Mathy[§] Panagiotis Papadimitriou[§]

[§]Computing Dept., Lancaster University, UK
{m.hoerdt, l.mathy, p.papadimitriou}@lancaster.ac.uk
[‡]Dept. of Computer Science, University College London, UK
{a.greenhalgh, m.handley}@cs.ucl.ac.uk
[*]NEC Europe, Heidelberg, Germany
felipe.huici@nw.neclab.eu

ABSTRACT

The Internet has seen a proliferation of specialized middlebox devices that carry out crucial network functionality such as load balancing, packet inspection or intrusion detection, amongst others. Traditionally, high performance network devices have been built on custom multi-core, specialized memory hierarchies, architectures which are well suited to packet processing. Recently, commodity PC hardware has experienced a move to multiple multi-core chips, as well as the routine inclusion of multiple memory hierarchies in the so-called NUMA architectures. While a PC architecture is obviously not specifically targeted to network applications, it nevertheless provides tremendous performance cheaply. Furthermore, a few commodity switch technologies have recently emerged offering the possibility to control the switching of flows in a rather fine grained manner. Put together, these new technologies offer a new network commodity platform enabling new flow processing and forwarding at an unprecedented flexibility and low cost.

1. INTRODUCTION

In the last few years two trends have started to reshape the Internet. The first of these is steady encroachment of economic reality on the architecture of the network itself; primarily this takes the form of embedding higher level knowledge *inside* the network to enhance the ability to manage services, make better use of limited resources and control costs. Usually this means using middleboxes such as firewalls[11], traffic shapers[14], load balancers[13], IDS and IPS systems[7], and application enhancement boxes[16, 17, 18]. It has become rare to find an end-to-end path that does not encounter at least one such device.

Middleboxes have become a multi-billion dollar market, but network operators do not spend all this money without good reason: such technologies have become essential to providing high levels of service for key applications. The great merit of the original Internet architecture was its ability to support as-yet-unforeseen applications, but the downside is that the network does not know when the applications it supports are actually working. To prosper, enterprises need additional control, and middleboxes provide this.

The second trend lies in the commoditization of hardware. Over many years components and systems designed primarily for the mass market achieve such large volumes of sales that their capabilities increase to displace high-end products. The canonical example is the rise of the Intel x86 CPU architecture, first displacing high-end Unix workstations running on RISC processors, and now breaking into the very top of the supercomputer league tables. In the data center, the commoditization of the 1U form-factor x86 server combined with drastically reduced CPU costs has greatly narrowed the price gap between server and desktop systems: a rack-mount case still costs more than a desktop case, but very capable servers can be bought for $1500 complete with multiple onboard Gigabit ethernet NICs.

It is, however, not only computers that have become commodity items. A few years ago, many people were forecasting the risk that network processors would displace custom silicon in high-end router platforms. However, something different happened. The combination of a rise in very capable and cheap chipsets for Gigabit ethernet from the likes of Broadcom and Marvell, huge volume shipments from companies such as Dell and Netgear, and bulk manufacturing from manufactures such as Quanta (who also make many of the world's laptops) has caused switching to become commoditized. As with x86 processors, the low end has started to increase in capability and displace high-end specialist products. Today's 48-port gigabit switches support both layer 2 and layer 3 forwarding, ACLs and other features at a price of around $20 per port; commodity 10 gigabit switches are now starting to emerge.

Researchers and middlebox manufacturers are both well aware of the capabilities of x86 commodity hardware, but the commoditization of switch hardware and the potential to rewrite their control software has not received quite the same attention. While switches have become more powerful, they are still relatively inflexible devices: as a platform, they are rather limited in capability. Things only become interesting when you combine switches with servers.

Consider now the confluence of these two trends. There is a huge proliferation of middleboxes, each servicing a single role performing L4-L7 functionality on data flows. At the same time, we now have cheap and extremely capable switching and processing components. However the switches are too dumb and the servers have their limitations (despite their pretty good performance, there is only so much you can do with one box before memory bottlenecks start to

kick in[6]). The clear solution is to build a generic network control, forwarding and flow processing platform from commodity switch hardware unified with a small cluster of servers, all managed as a single platform. Such a platform is inexpensive, very flexible, scalable, and tolerant of failure.

Perhaps more importantly, the rise of such platforms would open up the possibility of a commodity market for high-performance middlebox software, where a network operator might be able to mix and match control and management software in a way which is currently difficult at best.

The biggest downside of middleboxes is that they embed into the network knowledge of today's applications at the expense of tomorrow's innovations. It might seem like we are attempting to encourage this process, but the reality is that it has already happened. Once a middlebox is deployed, the cost of changing is substantial. We hope that by encouraging a common platform for such capabilities, and by making this market one for software rather than for appliances, the additional flexibility and reduced time to deployment might remove some of the barriers faced by innovative applications of the future.

The rest of the paper is organized as follows. Section 2 describes the building blocks or technologies we base the platform on in greater detail; section 3 provides an overview of *Flowstream*, our proposed flow processing platform, including usage scenarios and applications; section 4 discusses the consolidation of the platform, section 5 covers related work and section 6 highlights our conclusions.

2. BUILDING BLOCKS

So far we have identified two trends, the middleboxes in the network and the commoditization of servers and switches. We also reached the conclusion that the solution is to build a generic network control, forwarding and flow processing platform from these commoditized elements. In this section we describe the building blocks of such a platform in greater detail before discussing how these might be put together in the next section.

2.1 Commodity Switches

How cheap have network switches become? In order to answer this question, we conducted a survey of a range of lower-end gigabit switches, calculating for each the retail price per port. While this survey is by no means thorough, it gives a good idea of what the costs are when purchasing a lower-end network switch.

Figure 1 shows the results of the survey. We classified the switches into four groups: those with the simplest layer 2 forwarding (simple L2), those with layer 2 forwarding and advanced features such as ACLs (advanced L2), those with simple layer 3 forwarding capabilities (simple L3), and those with advanced layer 3 capabilities such as the ability to run a routing protocol (advanced L3). As can be seen, the prices range from about $10 to about $110 per port, putting the cost of a 48-port switch between $494 and $5,250, certainly within the price levels of what we would call commodity hardware.

Besides their cheap prices, many of these commodity switches share the same underlying chipsets. These chipsets are supplied by a small number of manufacturers, which is why many of the features of the switches are so similar among vendors. This is actually to our advantage: it would be possible to purchase kits from the manufactures to enable

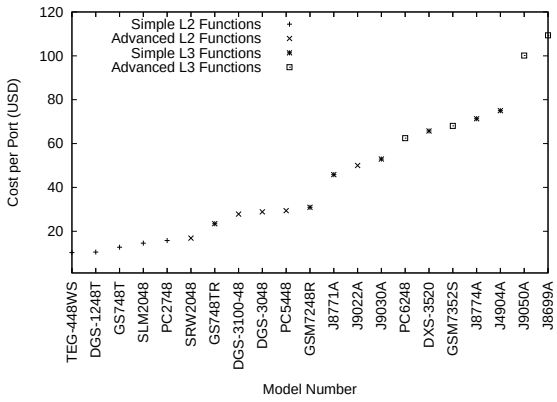


Figure 1: Cost per port for different switches.

the development and deployment of custom software for any proposed application.

2.1.1 Openflow

Modifying the embedded software system on switch hardware in order to perform custom operations is not for the faint-hearted, so ideally we would like to rely on switches with flexible software. Fortunately, Openflow[3] is an example of such a platform, with manufacturers like HP, NEC, Juniper and Cisco already producing prototype switches [12]. Openflow switches contain, among other things, a so-called *Flow Table* that can be configured by adding entries. These entries aggregate packets into flows by matching on a number of L2, L3 and L4 fields and then specify which port on the switch a particular flow should be sent out on, turning the switch into a simple yet rather flexible platform.

As mentioned, OpenFlow uses the flow as its basic unit of control, but is this the correct model for us? Traffic can be split at different levels, with the packet being the most common one. Despite giving fine-grained control, processing traffic at the packet level will almost certainly introduce packet reordering, something that we wish to avoid at all costs. Splitting traffic at the flow level would certainly prevent this, allowing us to send flows to different systems without having to worry about costly re-ordering operations. As a result, Openflow should provide the necessary level of control needed for our purposes. While at the time of the writing there were no such switches commercially available, we speculate that a chipset able to perform advanced L3 forwarding should be powerful enough to comply with the Openflow specification, giving an indication that the cost of these switches should also be reasonable.

2.2 Commodity PCs

Even though commodity PCs have been used for a while to process network traffic, it is only in recent years that improvements in various technologies have allowed them to become a powerful network platform. The introduction of PCI Express, for example, removed the bottleneck presented by its predecessor, PCI-X[10]. Further, the availability of an increasing number of CPU cores allows a PC to run several network processes concurrently while providing high performance to each of them (as long as memory hierarchy issues are carefully considered). Ethernet port density has also increased: quad-port cards are now commonplace, which com-

bined with motherboard interfaces allow a server to have as many as 15 or more ports.

The combination of these technologies with the drop in prices have rendered the PC a viable platform for network processing. Exactly how powerful can a commodity PC be? In previous work [6] we used a relatively inexpensive Dell 2950 server with 8 processors cores, 12 Ethernet ports, and standards-compliant IP forwarding paths implemented with Click Modular Router [9]. With this setup we were able to ip-forward most packet sizes at line rate, and minimum-sized packets at a very reasonable 4.9 million packets per second.

2.3 Virtualization and Virtual Routers

Virtualization techniques enable a PC to run multiple OSes (i.e., multiple instances of the same OS or optionally different OSes, depending on the virtualization technology used) concurrently, giving them access to the underlying hardware while isolating one from the other. In addition, virtualization makes it relatively easy to migrate these OSes to another PC, a mechanism that we will exploit later. Related work in [20] shows how to prevent network traffic disruption during the migration of virtual routers.

In [5] we tackled basic fairness issues and limitations of a modern PC for software packet forwarding, exploring alternative virtualization technologies and different forwarding scenarios. From these findings we designed a *virtual router* that has highly configurable forwarding planes for advanced programmability, optimized core scheduling for high performance, and hardware multi-queueing for sharing interfaces between virtual router instances. In [6] we analyzed the virtual router's performance, showing that virtual router solutions based on current commodity hardware represent a flexible, practical and inexpensive proposition.

3. FLOWSTREAM ARCHITECTURES

Given the building blocks described above, we can now discuss in more detail a class of systems architectures for building in-network processing platforms that we believe represents a "sweet spot" that nicely balances performance, scalability and flexibility. We call such platforms "Flowstream Architectures", for reasons that should be clear shortly. Platforms built according to the flowstream architecture can be characterized by the following properties:

- The core of the platform consists of an ethernet switch configured to route flows. A flow is defined in the OpenFlow sense, as packets that match a (possibly wildcarded) tuple of source and destination addresses and ports.
- Streams of data from these flows are then routed to one of a number of attached commodity server boxes for additional processing, before being forwarded on to the final destination.
- Software running on the server boxes can be composed to provide processing pipelines of modules.
- These modules are virtualized, in the sense that they can be moved between the servers to balance load and provide robust service in the presence of failures.
- The switch and servers are managed as a single platform from the point of view of the operators.

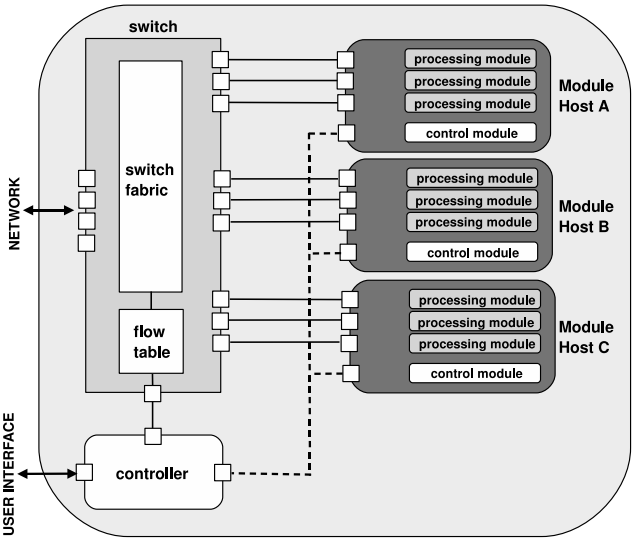


Figure 2: Overview of a Flowstream platform.

3.1 Description of a Platform

Figure 2 illustrates how the server boxes (we call them *module hosts* to distinguish them from traditional servers) and flow-based switch are connected together with a controller host to form a flowstream platform. Each host runs a number of *processing modules* where all of the actual flow processing takes place except for basic forwarding which can be done by the switch. Further, hosts contain a special module called a *control module*, which receives commands from the platform's controller to remove, install or migrate modules, as well as to provide monitoring information about the host's load and performance.

There are three main technologies available to us for implementing a module:

- A virtual machine running its own OS and module application.
- A process running on a virtual machine shared with other modules.
- A set of kernel forwarding elements instantiated in the kernel of the device driver domain on one of the module hosts.

The first of these options is the most general and provides the best inter-module isolation, whereas the third will provide the highest performance for traffic that needs to traverse several modules in the same module host. We envisage different applications will use different implementation options, often on the same flowstream platform.

For composing kernel forwarding elements, the Click modular router [9] provides a suitable set of building blocks. For example, a module can be composed of a predefined set of Click elements, and under the control of the operator, cascades of such modules can be plumbed together at run-time.

A Flowstream platform's second main component is the Openflow switch, providing the basic connectivity between module hosts and the network. In addition to this, the switch contains a flow table is configured from the controller at runtime, allowing different flows to be directed to any of the ports on the switch. It is worth pointing out that while

figure 2 shows a single switch, it would be certainly possible to scale the platform's port density by including additional switches.

The final component is the controller. Essentially, this is the brains of the platform and also its user interface to the outside world. When the operator makes a request (for instance, running an IDS on flows to a particular web server), the controller begins by choosing the module host or hosts to install the processing module(s) on. Such a decision could be based on the hosts' current load, information that the controller retrieves periodically from the control modules. Having selected a host, the controller then instructs the control module to install the requested processing module. Once this is done, the controller configures the switch's flow table so that the corresponding flows are directed to the right processing module.

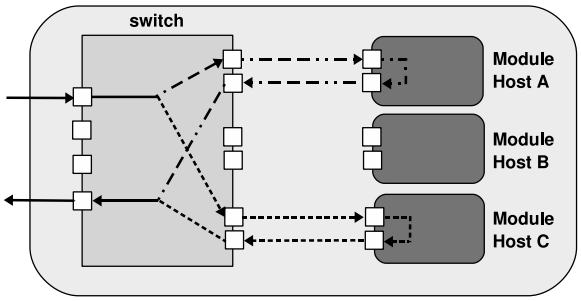
With all of these components in place, a flowstream architecture provides a powerful platform for flow processing. The fact that it is built upon commodity yet, as shown in previous work, high performance hardware should result in significant cost savings. In addition, a flowstream setup can be easily expanded and contracted dynamically by adding or removing module hosts, something that cannot be easily accomplished on conventional routers or middleboxes. Further, when required the isolation provided by virtualized module hosts means that several different flow processing operations can be performed simultaneously while minimizing negative interactions. The controller can migrate modules as required to ensure that a processing task does not significantly degrade the performance of others. Last but not least, using general-purpose processors and allowing operators to install their own flow processing modules yields great flexibility. So long as modules have access to well-defined flow APIs, a Flowstream platform can accommodate a wide range of existing and even future network applications. It is precisely the usage of the platform and its potential applications that we discuss next.

3.2 Usage Scenarios

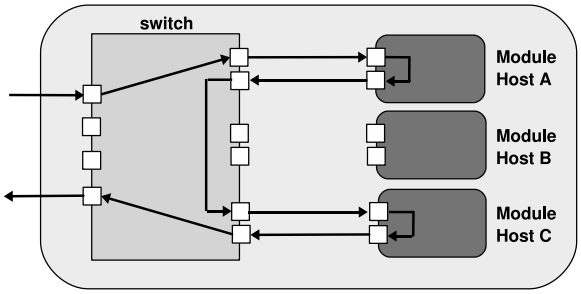
In the most basic case, the operator submits a request to a Flowstream platform's controller asking it to apply a certain processing module to a subset of the traffic being forwarded. The controller then chooses a module host with appropriate load levels and installs the module on it, then configures the switch's flow table. The flow then travels from the switch to the module for processing, before before being sent back to the switch and then out onto the network[1].

Beyond the simple case, there are two more interesting usage scenarios, depending on whether modules act on flows in parallel or serially. In parallel processing (see figure 3(a)), flows are load-balanced, pushing different flows to different module hosts but processing each of them equally. In this case identical processing modules run on multiple module hosts (in the figure, hosts A and C). The controller sets up the switch's flow table so that a flow gets sent to either of the module hosts, thus load balancing the traffic; an algorithm such as the RSS hash [4] can be used to accomplish this. To avoid reordering, all the packets from one flow must be processed by one module host. Flows could also be distributed unevenly based on the capabilities of the module hosts or

[1]Note that while so far we have described modules as receiving flows, processing them and then forwarding them, it is certainly possible for a module to act as a traffic sink



(a) Parallel processing (load-balancing) scenario.



(b) Serial processing scenario.

Figure 3: Basic platform usage scenarios.

their current load. Parallel modules are useful for quite a number of CPU-intensive network processing tasks, including intrusion detection, SPIT and DoS attack filtering, and monitoring and deep packet inspection.

In serial processing or pipelining (see figure 3(b)), the operations performed on flows are split across several module hosts and done one at a time. One example of an application for this is VPN termination, where one host could be used to perform the expensive encryption operation before another takes care of the tunneling. Serial processing is essential when each packet must be processed first by one module, then by another. Serial processing would also be useful if one of the hosts had dedicated hardware to perform an expensive operation at line rate, or if a module host did not have enough interfaces to carry out a particular function, such as acting as a router.

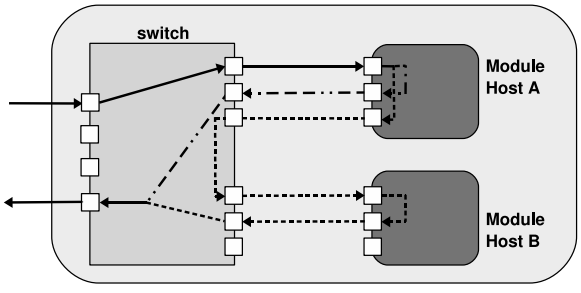


Figure 4: Scenario: offloading to a separate module host for further processing.

Combinations of serial and parallel are of course possible, as is heterogeneous parallel processing. In this case different flows are processed on different modules hosts, but they are also processed by different modules. For example, traffic to the web server farm might be processed by server load balancing modules, whereas traffic to the mail server may

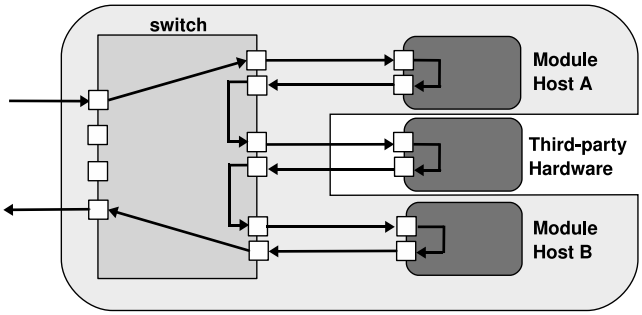


Figure 5: Inclusion of existing hardware

traverse a mail-server blacklist filter.

A more complex usage scenario is *flow splitting*, whereby a processing module is used to split a subset of traffic from a flow aggregate to another module for further processing (see figure 4). An application that fits rather well with this mechanism is intrusion detection: for example, module host A could be used to apply a quick, preliminary filter in order to separate out suspicious flows. Matching flows would then be sent to module host B for a more in-depth inspection, whereas those that do not match are sent back to the switch for immediate forwarding. It is worth pointing out that for simple filters, the actual splitting of flows could be done by the Openflow switch, thus off-loading some of the work from the module hosts.

Finally, because a flowstream platform consists of loosely-coupled hardware, it is possible to use it to encapsulate existing network middleboxes. For example, in figure 5 if a third party application accelerator box cannot cope with the full traffic rate, or needs some traffic excluded from its interference, such a box can be plugged into the flowstream switch and treated as a black box through which a subset of the traffic can be directed.

3.3 Module Migration

Flowstream architectures fit firmly into the trend of using arrays of cheap and potentially unreliable hardware, but providing robustness in software. To provide such robustness, we need to be able to migrate modules between hosts, both to manage changing load and to adapt to failures. All three of the mechanisms described for implementing modules can be migrated live between hosts, though with varying costs for the migration:

- Today's OS virtualization platforms can support live VM migration.
- Cluster computing platforms support live process migration.
- Click kernel forwarding paths can be reconfigured on the fly to include new elements.

It is perhaps this ability to migrage processing functions between hardware, while simultaneously re-plumbing the switch flow table to match, that perhaps best illustrates the flexibility of flowstream architectures. This flexibility can even be used to power down underused module hosts during quiet hours to save on electricity costs.

4. BENEFITS OF CONSOLIDATION.

A flowstream platform consolidates a number of middlebox systems into a single entity. For this to be worthwhile we need to gain tangible benefits from the consolidation; benefits that make the whole greater than its parts, otherwise we are just shifting functions from one system to another. This consolidation has the following benefits:

- Increased tolerance to failures.
- Reduced equipment and maintance costs.
- The ability to do dynamic reprovisioining.

Increased tolerance of failures can be achieved by having spare module hosts to take the place of a module host that has failed or is perceived to be about to fail. In a standard middlebox deployment each type of middlebox required a spare system to be available in case of hardware failure. In the flowstream architecture a smaller number of spare systems are required because module hosts are agnositic to the processing modules being run on them. Provided the hardware profile of the failed or failing module host is exceeded by the remaining spare capacity then we can distribute the processing modules from the failing module host on the spares. Module migration makes it possible to redistribute running modules, or as a last resort, to restart a module on a new host.

Reduced equipment and maintance costs are achieved by separating the logical and physical systems and adopting Google's model of using mass market commodity boxes. Expensive downtime is removed by migrating the processing modules to spare systems and then undertaking maintance on an offline system.

Dynamic reprovisioning is an outcome of the load balencing scenario presented in section 3.2. The flowstream architecture enables us, at a fairly fine granularity, to increase or decrease the capability of any processing module by varying the allocation of flows and processing resources with whole systems being shut down during quiet periods and brought back online when the load increases. New features can easily be trialed by splitting or copying a small portion of the traffic to the new processing module without interupting the live system.

5. RELATED WORK

In this section, we discuss related work and we position Flowstream among the very few related systems. [1] explores the scalability of software routers on general-purpose hardware, concentrating on particular scaling challenges, such as the per-packet processing capability in relation to the line rate. The authors eventually propose a clustered software-router architecture that uses an interconnect of multiple servers in order to enhance scalability for software routers. However, this architecture does not support the flow processing capabilities of Flowstream[2]. At the same time, Flowstream does not pose any considerable scalability issues (at least for the purpose it is designed for), since the platform's port density scales well with the addition of more OpenFlow switches.

[2]One flow processing module in Flowstream is IP forwarding, but this is not its main purpose

SuperCharging PlanetLab [19] advocates the decoupling of network nodes into a control/application plane running on commodity harware and specialized network-processor HW for the forwarding. In contrast, we advocate taking advantages of clusters of modern commodity hardware.

Pswitches [8] proposes the use of advanced commodity switches to control the paths of flows in datacenters, and share expensive middleboxes. Ethane [2] introduces the use of flow-based switching as a way to control and improve the security of enterprise network. An Ethane switch is essentially an early Openflow technology. Our platform takes the flexibility afforded by commodity switches further, by building complex network processing functionality within PC clusters.

In [15] the authors propose the combination of a programmable controler and switches for traffic management, while our approach considers the combination of commodity server and switching hardware to implement complex router applications beyong traffic management.

6. CONCLUSIONS

In this paper we have proposed a new class of system architectures for in-network processing platforms that has emerged from the confluence of the commoditization of switch and x86 server hardware and the arrival of capable open virtualization solutions. We have spelt out the architectural benefits and the sources of cost savings in such a class of processing platform. The interconnects for such an architecture are not yet defined, nor are the module APIs required to control and monitor a system based on this architecture. Openflow and the current open virtualization solutions are good starting points for this system but much remains to be done before a practical realization of this architecture is achieved with products from multiple vendors.

Overall, the emergence of the new commodity network hardware we propose is poised to speed up the convergence and integration of the network and data centers. Indeed, the software oriented network programmability that emerges would allow on the fly deployment of (new) network applications inside clouds of commodity computing. In essence, flowstream can be considered as taking further the separation and decoupling of network functionality and network infrastructure.

7. REFERENCES

[1] Katerina Argyraki, Salman Abdul Baset, Byung-Gon Chun, Kevin Fall, Gianluca Iannaccone, Allan Knies, Eddie Kohler, Maziar Manesh, Sergiu Nedveschi, and Sylvia Ratnasamy. Can software routers scale? In *Proceedings of PRESTO'08*, Seattle, USA, August 2008.

[2] Martin Casado, Michael Freedman, Justin Pettit, Nick McKeown, and Scott Shenker. Ethane: Taking control of the enterprise. In *Proceedings of SIGCOMM'07*, Kyoto, Japan, August 2007.

[3] Open Flow Switch Consortium. Open flow switch. `http://www.openflowswitch.org`.

[4] Microsoft Corporation. Scalable networking with RSS, white paper, November 2008.

[5] Norbert Egi, Adam Greenhalgh, Mark Handley, Mickael Hoerdt, Felipe Huici, and Laurent Mathy. Fairness issues in software virtual routers. In *Proceedings of PRESTO'08*, Seattle, USA, August 2008.

[6] Norbert Egi, Adam Greenhalgh, Mark Handley, Mickael Hoerdt, Felipe Huici, and Laurent Mathy. Towards high performance virtual routers on commodity hardware. In *Proceedings of ACM CoNEXT 2008*, Madrid, Spain, December 2008.

[7] Endace. Endace ninjabox network monitoring. `http://www.endace.com/ninjabox.html`.

[8] Dilip Joseph, Arsalan Tavakoli, and Ion Stoica. A policy-aware switching layer for data centers. In *Proceedings of ACM SIGCOMM 2008*, Seattle, USA, August 2008.

[9] Eddie Kohler, Robert Morris, Benjie Chen, John Jahnotti, and M. Frans Kasshoek. The click modular router. *ACM Transaction on Computer Systems*, 18(3):263–297, 2000.

[10] Jiuxing Liu, Mamidala A., Vishnu A., and Panda D.K. Performance evaluation of infiniband with pci express. In *High Performance Interconnects*, pages 13–19, San Diego, CA, USA, August 2004. IEEE.

[11] Check Point Software Technologies Ltd. Check point. `http://www.checkpoint.com/`.

[12] Nick McKeown. Enterprise GENI Talk, October 2008.

[13] F5 Networks. Big-ip product family. `http://www.f5.com/products/big-ip/`.

[14] Packeteer. Packeteer products. `http://www.packeteer.com/products/`.

[15] Hideyuki Shimonishi, Takashi Yoshikawa, and Atsushi Iwata. Off-the-path flow handling mechanism for high-speed and programmable traffic management. In *Proceedings of PRESTO'08*, Seattle, USA, August 2008.

[16] Citrix Systems. Citrix NetScaler.

[17] Citrix Systems. Citrix WANScaler.

[18] Riverbed Technology. Riverbed. `http://www.riverbed.com/`.

[19] Jon Turner, Patrick Crowley, John DeHart, Amy Freestone, Brandon Heller, Fred Kuhns, Sailesh Kumar, John Lockwood, Jing Lu, Michael Wilson, Charles Wiseman, and David Zar. Supercharging planetlab - a high performance, multi-application, overlay network platform. In *Proceedings of SIGCOMM'07*, Kyoto, Japan, August 2007.

[20] Yi Wang, Eric Keller, Brian Biskeborn, Jacobus van der Merwe, and Jennifer Rexford. Virtual routers on the move: Live router migration as a network-management primitive. In *Proceedings of SIGCOMM'08*, Seattle, USA, August 2008.