Prototyping Fast, Simple, Secure Switches for Ethane

Jianying Luo, Justin Pettit, Martin Casado, John Lockwood, Nick McKeown
Computer Systems Laboratory, Stanford University
Stanford, CA 94305, USA
{jyluo, jpettit, casado, jwlockwd, nickm}@stanford.edu

Abstract

We recently published our proposal for Ethane*: A clean-slate approach to managing and securing enterprise networks. The goal of Ethane is to make enterprise networks (e.g. networks in companies, universities, and home offices) much easier to manage. Ethane is built on the premise that the only way to manage and secure networks is to make sure we can identify the origin of all traffic, and hold someone (or some machine) accountable for it. So first, Ethane authenticates every human, computer and switch in the network, and tracks them at all times. Every packet can be immediately identified with its sender. Second, Ethane implements a network-wide policy language in terms of users, machines and services. Before a flow is allowed into the network, it is checked against the policy.*

Ethane requires two substantial changes to the network: Network switches and routers are replaced with much simpler switches, which are based on flow tables. The switch doesn't learn addresses, doesn't run spanning tree, routing protocols or any access control lists. All it does is permit or deny flows under the control of a central controller. The controller is the second big change. Each network contains a central controller that decides if a flow is to be allowed into the network. It makes its decisions based on a set of rules that make up a policy.

One premise of Ethane is that although the network is much more powerful as a whole, the switches are much simpler than conventional switches and routers. To explore whether this is true, we built 4-port Ethane switches in dedicated hardware (on the NetFPGA platform), running at 1Gb/s per port. We have deployed the switches in our network at Stanford University, and demonstrated that despite the simplicity of the switches, Ethane can support a very feature-rich and easy-to-manage network.

1. Introduction

1.1. Background

Security and management are increasingly important in enterprise networks. A Yankee Group report found that 62% of network downtime in multi-vendor networks comes from human-error, and that 80% of IT budgets is spent on maintenance and operations [7]. Network intrusions routinely lead to costly down-times [1] and the loss of intellectual property [5].

Industry has responded by building switches and routers that support increasingly complex functions [12, 4] such as filtering using ACLs, VLANs, NAT and deep-packet classification. Through the introduction of middle-boxes, increasing layers of band-aids are being added to the network—each one adding another layer of complexity that makes the network harder to manage. These boxes typically have to be placed in choke points (to make sure all the traffic passes through them), making the networking less scalable.

We believe that if we continue to add more and more complexity in the network, it will make it less scalable, more power-hungry, harder to manage, less reliable and ultimately less secure. Instead, we believe the answer lies in removing complexity from the switches and routers, reducing them to the very simplest forwarding devices. We believe that the decision of who can communicate with whom should be managed by a network-wide policy running in software in a central location. As we hope to show, new high-performance computers are now fast enough to make this practical for quite large networks. This way, networks can be managed, controlled and secured more easily, and can evolve more rapidly.

We call our approach Ethane, a new enterprise security architecture based on Ethernet. In [3] we described Ethane, as well as a prototype network that we built and deployed in the Gates Computer Science building at Stanford University.

Ethane has two main characteristics: First, it always knows who sent a packet. By keeping track of the loca-

tion of all users and machines, and authenticating all names and addresses, Ethane can always associate a packet with a particular user or host. Second, Ethane maintains a central policy (declared at a central controller) to decide who can communicate with whom. Whenever a flow starts, Ethane checks the first packet against the network policy: If the flow is allowed, the Controller selects a route for the flow and adds a flow-entry to all the switches along the path. If the flow is denied, no flow-entry is added and its packets are not forwarded.

An Ethane network has two new parts: (1) a *central controller* running in software. The Controller checks all new flows against a network-wide security policy, picks the route for accepted flows and installs a flow-entry in every switch along the path. The Controller handles all bindings (it keeps track of which port machines are connected to, it hands out all IP addresses, it handles DNS requests, and it authenticates all users, end-hosts and Switches); and (2) Ethernet switches (and routers) are replaced with *Ethane Switches*. An Ethane Switch consists of a flow table (one entry per authorized flow), and a secure connection to the Controller. The first packet of every new flow is sent to the Controller; if the Controller accepts the flow, it installs a new flow-entry in the Switch.

Our prototype network ran for three months with over 300 end-hosts and 19 Switches. We deployed a Controller (running on a desktop PC), and several types of Ethane Switch: Wireless access points, 4-port Gigabit Ethernet hardware Switches (running on the NetFPGA platform), and 4-port software Switches (based on a desktop CPU and a 4-port NIC). More details of our experimental network and our experience deploying it can be found in [3].

1.2. This paper

One of the claimed advantages of Ethane is that the Switches are very simple: They contain only a flow table, and they don't do learning, MAC or IP address lookup, NAT, ACLs, routing, spanning tree, or source address filtering. Not only does this make them smaller, lower-cost and consume less power, it means that—by moving the complexity to the central Controller—it is easier to evolve the network and add more functionality in a single location.

In this paper we describe the implementation of the Ethane Switch in hardware. Although it is infeasible for us to develop custom ASICs, we can use the NetFPGA platform to build a 4-port Gigabit Ethernet-based Ethane Switch. The Switch is implemented in Verilog using an industry-standard design flow, allowing us to estimate the gate and memory count needed to implement a real Switch—probably with a larger number of ports—in a current ASIC process.

In what follows, we provide an overview of the opera-
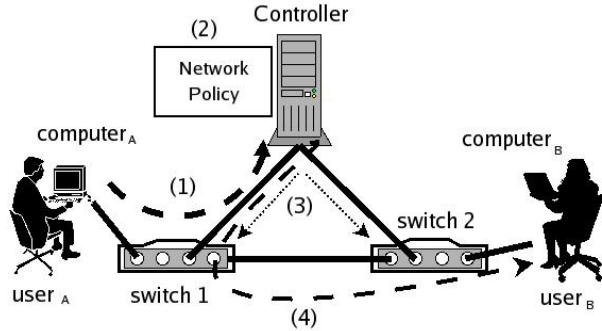
Figure 1. Example of communication on an Ethane network

tion of an Ethane network in §2. §3 describes the design and implementation of our Ethane Switch in hardware. We present a performance analysis of our hardware implementation in §4. Finally we present related work in §5, and our conclusions in §6.

2. Overview of Ethane

Figure 1 shows a simple Ethane network.[1]

When a packet arrives, an Ethane Switch checks the packet against its flow table. If it matches an entry, the Switch sends the packet to the specified outgoing port; otherwise, it forwards it to the Controller (Step 1).

When the Controller receives a packet from a Switch, it checks it against the global policy (Step 2). If it's allowed, the Controller computes the route[2] (enforcing any constraints defined in the policy, such as directing specific traffic through waypoints) and adds a new entry to every flow table along the path (Step 3). The Controller sends the packet back to the Switch, which forwards it along the new path (Step 4). Flow entries are removed after an inactivity period or if revoked by the Controller.

While basic Ethane Switches only contain a flow table, one can imagine slightly more sophisticated Switches with more capabilities (although we need to guard against the slippery slope that made switches as complex as they are today). For example, a Switch might contain multiple priority queues per port; and the Controller could decide which queue a flow belongs to. Or a Switch could rewrite the header as directed to by the Controller; e.g. to implement NAT (or a new variant), or to change MAC addresses at each hop to confuse eavesdroppers.

Many natural questions arise from using a central Controller: What are the performance requirements of the Con-

---

[1]For a more comprehensive description of Ethane, see [3].

[2]The Controller constructs the full network topology from link-state information sent by each Switch.

troller? What happens when a Switch or the Controller fails? How does the Controller protect itself from attacks? What happens when users move? A complete discussion of these topics falls outside the scope of this paper, so we refer interested readers to [3]. However, our testing indicates that a single Controller can comfortably handle over 10,000 new flow requests per second—enough to handle a 22,000-host network.

3. Switch Design

3.1. Overview

The Switch datapath is essentially a managed flow table. Flow entries contain a *Header* (to match packets against), an *Action* (to tell the Switch what to do with the packet), and *Per-Flow Data* (which we describe below).

There are two common types of entry in the flow table: per-flow entries for application flows that should be forwarded, and per-host entries that describe misbehaving hosts whose packets should be dropped. For TCP/UDP flows, the Header field covers the TCP/UDP, IP, and Ethernet headers, as well as physical port information. The associated Action is to forward the packet to a particular interface, update a packet-and-byte counter (in the Per-Flow Data), and set an activity bit (so that inactive entries can be timed-out). For misbehaving hosts, the Header field contains an Ethernet source address and the physical ingress port. The associated Action is to drop the packet, update a packet-and-byte counter, and set an activity bit (to tell when the host has stopped sending). Only the Controller can add entries to the flow table. Entries are removed because they timeout due to inactivity (local decision) or because they are revoked by the Controller. The Controller can revoke a single, badly behaved flow, or a whole group of flows belonging to a misbehaving host, a host that has just left the network, or a host whose privileges have just changed. The flow table is implemented using two exact-match tables: One for application flow entries and one for misbehaving host entries. Because flow entries are exact matches, rather than longest-prefix matches, it is easy to use hashing schemes in conventional memories rather than expensive, power-hungry TCAMs.

Other Actions are possible in addition to just forward and drop. For example, a Switch might maintain multiple queues for different classes of traffic, and the Controller can tell it to queue packets from application flows in a particular queue by inserting queue IDs into the flow table. This can be used for end-to-end L2 isolation for classes of users or hosts. A Switch could also perform address translation by replacing packet headers. This could be used to obfuscate addresses in the network by swapping addresses at each Switch along the path (an eavesdropper would not be able
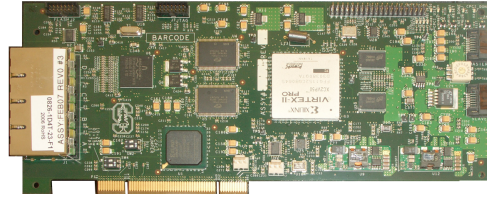
Figure 2. Photograph of the NetFPGA circuit board

to tell which end-hosts are communicating) or to implement address translation for NAT in order to conserve addresses. A Switch could also control the rate of a flow.

The Switch also maintains a handful of implementation-specific entries to reduce the amount of traffic sent to the Controller. This number should remain small to keep the Switch simple, although this is at the discretion of the designer. On one hand, such entries can reduce the amount of traffic sent to the Controller; on the other hand, any traffic that misses on the flow table will be sent to the Controller anyway, so this is just an optimization.

The Switch needs a small local manager—running in software—to establish and maintain the secure channel to the Controller, to monitor link status, and to provide an interface for any additional Switch-specific management and diagnostics. The software also handles hardware exceptions (e.g. flow table overflow).

3.2. Hardware Forwarding Path

The job of the hardware datapath is to process as large a fraction of packets as possible, and leave relatively few to the slower software. An arriving packet is compared against the flow table. If it matches, the associated Action is executed (e.g. forward, drop, over-write header). If it doesn't match, the packet is sent to software.

Our Switch hardware is built on the NetFPGA platform [10] developed in our group at Stanford University. A NetFPGA card plugs into the PCI slot of a desktop PC, and the Switch software runs in Linux on the desktop PC. A NetFPGA board (shown in Figure 2) consists of four Gigabet Ethernet interfaces, a Xilinx Virtex-Pro-50 FPGA, memory (SRAM and DRAM) and a PCI interface to the host computer. NetFPGA is designed as an open platform for research and teaching using an industry-standard design flow. Packets can be processed at full line-rate in hardware.[3]

In our hardware forwarding path, packets flow through a standard pipeline of modules. All the modules run in parallel and complete in 16 clock cycles (at 62.5MHz). The

---

[3]The implementation described here is running on version 2.0 of NetFPGA, and is currently being ported to version 2.1, which runs more than twice as fast. The functional Switch on version 2.1 of NetFPGA will be available in summer 2007.

modules check packet lengths, parse packet headers, implement hash functions, perform lookup functions, track traffic statistics, maintain a flow table in SRAM, and enable overwrite of packet headers as needed. The main design choice is how to implement the flow table.

We chose to implement the flow table as hash table (made easy because the flow table requires only exact-match lookups). In particular, we use double-hashing: we use two CRC functions to generate two pseudo-random indices for every packet—each index is used to lookup into a different hash table. This way, we make it very likely that at least one of the indices will find a unique match. If both hashes return entries that clash with existing entries, we say there has been a collision, and rely on software to process the packet.

A block level diagram of the Ethane datapath is illustrated in Figure 3.
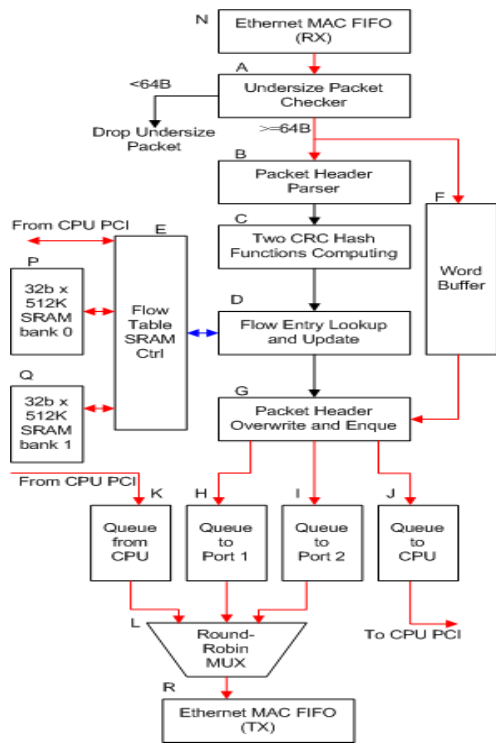
Figure 3. Block diagram of Ethane datapath

3.3. Modules in the Datapath

In Block A, received packets are checked for a valid length, and undersized packets are dropped.

In preparation for calculating the hash-functions, Block B parses the packet header to extract the following fields: Ethernet header, IP header, and TCP or UDP header. A *flow-tuple* is built for each received packet; for an IPv4 packet, the tuple has 155 bits consisting of: MAC DA (lower 16-bits), MAC SA (lower 16-bits), Ethertype (16-bits), IP src address (32-bits), IP dst address (32-bits), IP protocol field (8-bits), TCP or UDP src port number (16-bits), TCP or UDP dst port number (16-bits), received physical port number (3-bits).

Block C computes two hash functions on the flow-tuple (padded to 160-bits), and returns two indices; Block D uses the indices to lookup into two hash tables in SRAM. In our design, we use a single SRAM to hold both tables, and so have to perform both lookups sequentially.[4] The flow table stores 8,192 flow entries. Each flow entry holds the 155-bit flow tuple (to confirm a hit or a miss on the hash table), and an 152-bit field used to store parameters for an action when there is a lookup hit. The action fields include one bit to indicate a valid flow entry, three bits to identify a destination port (physical output port, port to CPU, or null port that drops the packet), 48-bit overwrite MAC DA, 48-bit overwrite MAC SA, a 20-bit packet counter, and a 32-bit byte counter.

Block E controls the SRAM, arbitrating access for two requestors: The flow table lookup (two accesses per packet, plus statistics counter updates), and the CPU via the PCI bus. Every 16 system clock cycles, the module can read two flow-tuples, update a statistics counter entry, and perform one CPU access to write or read 4 bytes of data. To prevent counters from overflowing, the byte counters need to be read every 30 seconds by the CPU, and the packet counters every 0.5 seconds (in our next design, we will increase the size of the counter field to reduce the load on the CPU, or use well-known counter-caching techniques, such as [9]).

The 307-bit flow-entry is stored across two banks of SRAM. Although we have 4MB of SRAM, our current design only uses 320KB, which means our table can hold 8,192 entries. It is still too early to tell how large the flow table needs to be—our prototype network suggests that we can expect only a small number of entries to be active at any one time. However, a modern ASIC could easily embed a much larger flow table, with tens of thousands of entries, giving headroom for situations when larger tables might be needed; e.g. at the center of a large network, or when there is a flood of broadcast traffic.

Block F buffers packets while the header is processed in Blocks A-E. If there was a hit on the flow table, the packet is forwarded accordingly to the correct outgoing port, the CPU port, or could be actively dropped. If there was a miss on the flow table, the packet is forwarded to the CPU. Block G can also overwrite a packet header if the flow table so indicates.

Overall, the hardware is controlled by the CPU via memory-mapped registers over the PCI bus. Packets are

---

[4]A higher performance Switch would, presumably, use two or more memories in parallel if needed.

transferred using standard DMA.

3.4. Software Control and Management

The Ethane Switch software has two responsibilities: First, it establishes and maintains a secure channel to the Controller. On startup, all the Switches find a path to the Controller by building a modified spanning-tree, with the Controller as root. The control software then creates an encrypted TCP connection to the Controller. This connection is used to pass link-state information (which is aggregated to form the network topology) and all packets requiring permission checks to the Controller. Second, the software maintains a flow table for flow entries not processed in hardware, such as overflow entries due to collisions in the hardware hash table, and entries with wildcard fields. Wildcards are used for the small implementation-specific table, and in our design are used for control traffic with loose speed requirements. The software also manages the addition, deletion, and timing-out of entries in the hardware.

If a packet doesn't match a flow entry in the hardware flow table, it is passed to software. The packet didn't match the hardware flow table because: *(i)* It is the first packet of a flow and the Controller has not yet granted it access *(ii)* It is from a revoked flow or one that was not granted access *(iii)* It is part of a permitted flow but the entry collided with existing entries and must be managed in software *(iv)* It matches a flow entry containing a wildcard field and is handled in software.

In our software design we maintain two flow tables; one is a secondary hash table for implementation-specific entries—as an optimization to reduce traffic to the Controller. For example, we used the second table to set up symmetric entries for flows that are allowed to be outgoing-only. Because we can't predict the return source MAC address (when proxy ARP is used), we save a lot of traffic to the Controller if we maintain entries with wildcards for the source MAC address and incoming port. Because it is in software, we made the second flow table large—32,768 entries. In practice, as we will see, a much smaller table would suffice. The other flow table is a small (1500-entry) associative memory to hold flow-entries that could not find an open slot in either of the two hash tables. In a dedicated hardware solution, this small associative memory would be placed in hardware. Alternatively, a hardware design could use a TCAM for the whole flow table in hardware.

4. Results

Now that we have a hardware implementation of an Ethane Switch deployed in our prototype network, we can ask several performance questions. We are most interested in three questions: *How big does the flow table need to be*

for reasonable performance?, *How complicated is the hardware of an Ethane Switch, particularly when compared to a regular Ethernet switch?*, and *Can our NetFPGA design process minimum length packets at the Gigabit Ethernet line-rate?*

4.1. How big does the flow table need to be?

In an Ethernet switch, the forwarding table is sized to minimize broadcast traffic: as switches flood during learning, this can swamp links and makes the network less secure.[5] As a result, an Ethernet switch needs to remember all the addresses it's likely to encounter; even small wiring closet switches typically contain a million entries in a large, power-hungry off-chip memory (SRAM or TCAM).

Ethane Switches, on the other hand only need to keep track of flows in-progress. For a wiring closet, this is likely to be a few hundred entries at a time, small enough to be held in a tiny fraction of a switching chip. Even for a campus-level Switch, where perhaps tens of thousands of flows could be ongoing, it can still use on-chip memory that saves cost and power.

To demonstrate, we have analyzed traces from our local Ethane deployment of 300 nodes as well as public traces collected from LBL [8]. Figure 4 shows the number of active flows at a switch in our Ethane deployment over 10 days—it never exceeded 500. With a table of 8,192 entries and a two-function hash-table, we never encountered a collision. A dataset collected from LBL (Figure 5) has a maximum of 1500 simultaneous flows within the 8,000 host network.

From these numbers we conclude that a Switch for a university-sized network should have a flow table capable of holding 8-16K entries. If we assume that each entry is 64B, the table requires about 1MB to 4MB if using a two-way hashing scheme [2]. In contrast, a typical commercial enterprise Ethernet switch today holds 1 million Ethernet addresses (6MB, and larger if hashing is used), 1 million IP addresses (4MB of TCAM), 1-2 million counters (8MB of fast SRAM), and several thousand ACLs (more TCAM). We conclude that the memory requirements of an Ethane switch are quite modest in comparison to today's Ethernet switches.

4.2. How complicated is the hardware?

We measured the NetFPGA resources used by our Ethane hardware design. In our count, we exclude the space required to instantiate the Ethernet MACs[6] on the Xil-

---

[5]In fact, network administrators often use manually configured and inflexible VLANs to reduce flooding.
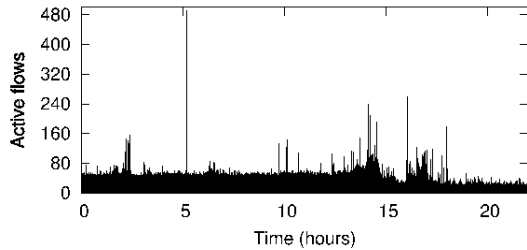
[6]They are tri-mode GE, 100Mb/s, 10Mb/s MACs.

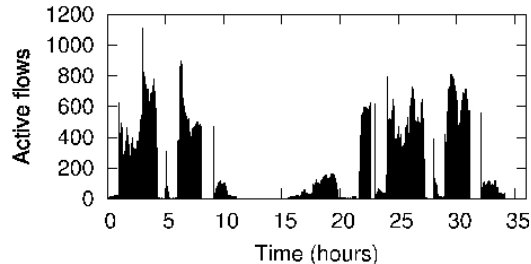**Figure 4. Active flows through two of our deployed switches**

Figure 5. Active flows for LBL network [8]

Packet Size	64 bytes	65 bytes	100 bytes	1518 bytes
Measured	1524Mbps	1529Mbps	1667Mbps	1974Mbps
Optimal	1524Mbps	1529Mbps	1667Mbps	1974Mbps

Table 1. Hardware forwarding speeds for different packet sizes. All tests were run with full-duplex traffic. Totals include Ethernet CRC, but not the inter-packet gap or the packet preamble. Tested with an Ixia 1600T traffic generator.

inx Virtex-II Pro FPGA because these circuits need to be present for any switch design.

In total, our Ethane Switch (without the MACs) utilized 6,142 of total 27,392 Slice Flip Flops, 8,790 of total 27,392 4-input LUTs, and 49 of total 136 Block RAMs each with 18Kb capacity. These utilized resources account for 32.1% of logic cells and 36.0% of Block RAMs in a Virtex II-Pro 30 chip. When implemented on the Virtex-II Pro 50 FPGA chip on our newer platform, the datapath (excluding the MACs) uses 18.6% of the available logic cells and 21.1% of Block RAMs. The 8,192-entry flow table in the datapath design uses 320KB of the external SRAM.

We also synthesized our hardware using a commercial 65nm ASIC standard cell library. The Ethane hardware logic, excluding memories and MACs, uses 61,971 gate-equivalents and would occupy just 0.1289 mm^2 of die. The 8,192-entry flow table—if built from on-chip SRAM—would take up 1.8 mm^2. In comparison, a typical modern Ethernet switch implemented in a 65 nm ASIC would require millions of gates, its switching table of 1 million entries an area of about 39 mm^2, and its routing table with 250K entries about 37 mm^2. As a packet processing datapath, the Ethane hardware would use over 30 times less area than a typical modern Ethernet switch.

4.3. Can our NetFPGA implementation process packets at line-rate?

The simple answer is yes. Table 1 shows the forwarding speed of our design for different packet sizes. Because of the required inter-packet gap and preamble, the maximum data rate is reduced for short packets. As can be seen from the table, our design achieves full line-rate for Gigabit Ethernet for all packet sizes.

For comparison, we repeated the tests with a software-only Switch. The Switch ran on a 3.2GHz Intel Celeron CPU with a 4-port Gigabit Ethernet PCIe network interface card. Packet processing was placed in the kernel to maximize throughput. For 1518 bytes packets, the software Switch sustained 1Gb/s. But as we reduced the packet size, the CPU could not keep up. At 100 bytes, the Switch could only achieve a throughput of 16 Mb/s[7]. Clearly, for now, the switch needs to be implemented in hardware.

5. Related Work

In 4D [6], a centralized control plane pushes forwarding tables and filters into the network. Our work extends this approach by providing fine-grained control and complex forwarding functions defined and enforced per flow.

SANE [4], a previous system we designed, used encrypted source routes assigned by a central controller to enforce a network-wide policy. Unlike Ethane, Sane required modifications to all end-hosts as well as the complete network infrastructure.

The Shunt project [11] also proposes a flow-based approach to enforcing network security policy from a single network appliance. However, Ethane does not require a single choke point to exact control over the network since it subsumes both security policy *and* network forwarding. Additionally, Ethane provides rich forwarding mechanisms such as isolation and NATing.

6. Conclusion

Several factors suggest that there will be a trend away from adding more complexity on the datapath of Switches and routers—and there might be a tendency towards simplicity and streamlining. Obvious reasons include the desire

---

[7]The drastic degradation in performance appears to be partially due to interrupt handling overheads. Better throughput could likely be achieved via polling in the network driver.

for higher port density, lower cost and lower power consumption. We anticipate that the need for reliability and manageability will drive down complexity; and the desire to make networks easier to evolve—coupled with higher performance computers - will lead to functionality being moved from the datapath to central controllers running in software. Last, but perhaps first, the increasing attention paid to security (and accountability) suggests that network functions will be brought together in a single controller, including routing, topology management, address allocation and DNS.

Ethane is our proposal for moving complexity off the datapath, and to centralize the management and control of networks.

The success of Ethane does not particularly depend on the Switches being simpler and lower cost—we believe the main advantage is the control it gives to the network administrator. However, it is still worth understanding how simple the Ethane Switches will be. Our results presented here suggest that an Ethane Switch can be considerably simpler than a conventional Ethernet switch or router.

7. Acknowledgments

References

[1] Sasser worms continue to threaten corporate productivity. http://www.esecurityplanet.com/alerts/article.php/3349321, 2004.

[2] A. Z. Broder and M. Mitzenmacher. Using multiple hash functions to improve IP lookups. In *Proc. INFOCOM*, Apr. 2001.

[3] M. Casado, M. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking control of the enterprise. In *SIGCOMM Computer Comm. Rev.*, Aug. 2007.

[4] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, N. McKeown, and S. Shenker. SANE: A protection architecture for enterprise networks. In *USENIX Security Symposium*, 2006.

[5] D. Cullen. Half life 2 leak means no launch for Christmas. http://www.theregister.co.uk/2003/10/07/half_life_2_leak_means/, Oct. 2003.

[6] 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. In *SIGCOMM Computer Comm. Rev.*, Oct. 2005.

[7] Z. Kerravala. Configuration management delivers business resiliency. *The Yankee Group*, Nov. 2002.

[8] R. Pang, M. Allman, M. Bennett, J. Lee, V. Paxson, and B. Tierney. A first look at modern enterprise traffic. In *Proc. Internet Measurement Conference*, Oct. 2005.

[9] D. Shah, S. Iyer, B. Prabhakar, and N. McKeown. Maintaining statistics counters in line cards. In *IEEE Micro*, Jan-Feb 2002.

[10] G. Watson, N. McKeown, and M. Casado. NetFPGA: A tool for network research and education. In *Workshop on Architecture Research using FPGA Platforms*, Feb. 2006.

[11] N. Weaver, V. Paxson, and J. M. Gonzalez. The shunt: an FPGA-based accelerator for network intrusion prevention. In *FPGA '07: Proceedings of the 2007 ACM/SIGDA 15th international symposium on Field programmable gate arrays*, pages 199–206, 2007.

[12] G. Xie, J. Zhan, D. A. Maltz, H. Zhang, A. Greenberg, and G. Hjalmtysson. Routing design in operational networks: A look from the inside. In *Proc. SIGCOMM*, Sept. 2004.

---

[8]Disclaimer: Any opinions, findings, conclusions, or recommendations expressed in this article are those of the authors and do not necessarily represent the views of the sponsors supporting this project.