Towards a Cost-Effective Networking Testbed

Nikola Knežević, Simon Schubert and Dejan Kostić
School of Computer and Communication Sciences, EPFL, Switzerland
email: firstname.lastname@epfl.ch

ABSTRACT

The Internet is suffering from ossification. There has been substantial research on improving current protocols, but the vendors are reluctant to deploy new ones. We believe that this is in part due to the difficulty of evaluating protocols under realistic conditions. Recent wide-area testbeds can help alleviate this problem, but they require substantial resources (equipment, bandwidth) from each participant, and they have difficulty in providing repeatability and full control over the experiments. Existing in-house networking testbeds are capable of running controlled, repeatable experiments, but are typically small-scale (due to various overheads), limited in features, or expensive.

The premise of our work is that it is possible to leverage the recent increases in computational power to improve the researchers' ability to experiment with new protocols in lab settings. We propose a cost-effective testbed, called MX, which emulates many programmable routers running over a realistic topology on multi-core commodity servers. We leverage open source implementations of programmable routers, such as Click, and modify them to allow coexistence of multiple instances in the same kernel in an effort to reduce packet forwarding overheads. Our initial results show that we outperform similar cost-effective solutions by a factor of 2. Next, we demonstrate that grouping and placing routers on to cores which share the L2 cache yields high performance.

1. INTRODUCTION

The Internet is suffering from ossification. For example, despite numerous proposals for improving the BGP convergence properties, these protocols have not been deployed. Hence, there still exist in the Internet end-to-end connectivity problems that last for a few minutes. We believe that this is in part due to the difficulty of evaluating new proposals under realistic conditions and convincing router vendors to deploy them.

As the bandwidth-delay products of the Internet links keep increasing, issues with TCP's convergence time, throughput, and amount of queuing in these environments are becoming more pressing. A load-factor based congestion control approach holds great promise as it requires moderate changes in routers (only to monitor and insert current load data into packets) and endpoints (to use a congestion controller that uses router feedback). This is just one example of a next-generation protocol that requires router support. Unfortunately changing the routers is difficult, as it typically requires convincing the vendor to incorporate the new features. Without a large set of users that demand new features, the vendor is not willing to commit to the engineering effort, thereby closing the vicious circle. We believe that the answer is in providing the networking researchers the ability to subject the new protocols to realistic network conditions (bandwidth, latency, losses, and even new hardware characteristics) in controlled, repeatable experiments.

Recent work on wide-area testbeds [4] can help meet this goal, but this typically requires substantial resources (equipment, bandwidth) for obtaining the rights to use the testbed. Further, while excellent for exposing the protocol to real traffic, unexpected conditions, and failure scenarios (thus making them irreplaceable in the last phases of protocol testing), live testbeds make it more difficult to distinguish between real protocol issues and experimental noise. In contrast, network simulators (e.g., ns2) offer a higher degree of control, but they can miss important system interactions. In addition, they do not allow for the direct execution of software prototypes. As a "middle ground" between simulation and wide-area testbeds, scalable emulation testbeds can serve a crucial role in fulfilling the researchers' needs. In such an environment, it is possible to emulate a network in which only some software routers are running a modified version of the control or data plane, and to examine global behavior when the network is subjected to "destructive" faultloads, such as partial and network-wide upgrades. Existing networking testbeds are capable of running controlled, repeatable experiments, but are typically small-scale (due to various overheads), limited in features, or expensive.

The premise of our work is that it is possible to leverage the recent increases in computational power to develop a cost-effective emulation testbed which can enable the desired controlled, high-fidelity network experiments. In this paper, we describe the design MX, a testbed which can run many programmable routers over an emulated topology using multi-core machines. MX runs live, unmodified protocols implemented in XORP [10] and Click [14] over Internet-like topologies.

2. MX DESIGN AND IMPLEMENTATION

In the remainder of this section we provide an overview of MX, followed by a description of the key techniques we use to address the challenges in meeting the requirements for networking testbeds.

2.1 Overview

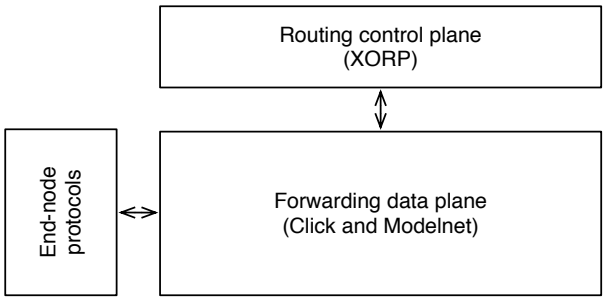

Figure 1: Overview of MX components

Figure 1 outlines the three main components of the MX testbed, each of which can run on separate machines.

In the control (routing) plane, we use XORP [10], an extensible routing platform. Since this component is only responsible for routing table computations (and no forwarding), we can collocate several XORP instances on a single machine.

The data plane consists of modified versions of Click [14] and ModelNet [15]. ModelNet is responsible for emulating network links, modeling losses, bandwidth, and delays.

We have observed that the router thread placement on the CPU cores affects performance. Thus, an important challenge is to design and implement a *core allocator*. This component of the data forwarding plane should use the emulated link traffic statistics and the CPU statistics to determine the thread-to-core mapping for high-throughput. Whenever the core allocator determines that the system will benefit from running a specific Click router on a different core, it moves (and pins) the router to the new core. When the traffic patterns change, the core allocator reassesses the router allocation, and possibly reassigns some routers to different cores. Figure 2 shows one possible assignment of routers to CPU cores over a sample topology. The core allocator does not the alter the emulated network topology in any way; it merely changes the mapping of Click routers to CPU cores. We are currently implementing this component.

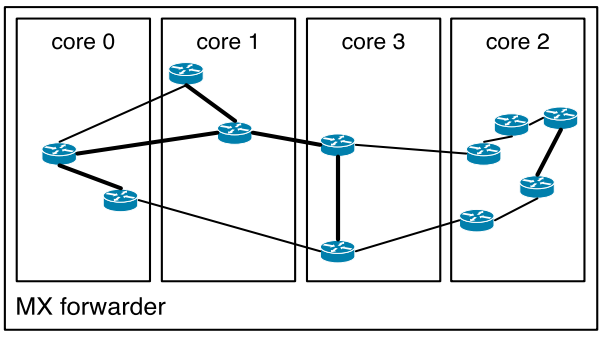

Figure 2: An example showing possible allocation of routers to CPU cores in MX.

The third MX component comprises the end-host (edge) plane where we can run different end-host protocols. We adopt the approach used in ModelNet, where one or more application instances can be multiplexed on a single machine.

2.2 Implementation Highlights

We took several steps in modifying Click to enable it to support multiple instances running on the same physical hardware. First, `FromDevice`/`ToDevice` are now taking packets from a ModelNet queue, removing the need for Click to access the actual NIC. Second, we removed all global objects, and had them encapsulated in router instances. As Click uses a concept of handlers to interact with the environment via the filesystem interface, we modified the handler handling code to work on instances (rather than on the single global object). Now, each virtual router has its own filesystem subtree. Finally, we modified Click so it runs in a kernel thread, instead of the net interrupt callout.

We modified XORP so as to not install any route on the host machine. Instead, the XORP instances transmit the routing changes via a dedicated channel to the data plane.

2.3 Meeting the Testbed Requirements

As the research community moves toward the future Internet, it needs networking testbeds that will let researchers try out their ideas. An ideal networking testbed should satisfy several, sometimes conflicting, goals. Here we discuss these requirements and outline how MX tries to meet them.

Ability to run standard protocol implementations. A testbed should be able to run the same (or slightly modified) version of a protocol as if used in the real environment. This approach ensures that observations made in the test run correspond to reality, thus increasing vendor confidence in the protocol implementation.

MX supports standard protocol implementations (OSI level 3 and higher) by using XORP and Click as development platforms for the routing and the forwarding plane, respectively. Since both of these platforms are extensible, MX shares the same feature. All the changes we made to Click and XORP are invisible to the end user. Moreover, MX supports testing of user-level application protocols, as all traffic from end-node machines (Figure 1) is sent through the core.

Full control of the experiments/repeatability. Certain failure scenarios can occur rarely in live deployments. Thus, the testbed should offer full control over the topology characteristics and faultloads used in the experimentation. Doing so can significantly speed up the development process. Another related point is repeatability – the testbed should allow the researcher to run the same experiment several times under identical conditions. This requirement is especially important during debugging, and is a basic requirement for doing scientific work. If the conditions are changing from one experiment to another, the researcher will have difficulty in separating the behavior of the protocol from that of the environment.

With the full control of the testbed hardware and of the code running in all three planes (control, data and application plane), there is a high amount of repeatability in the MX experiments. However, we cannot guarantee complete repeatability, since there may be losses and timing issues, i.e. during a specific experiment a packet P_1 can arrive at router R_1 before packet P_2 reaches router R_2, and vice-versa during another execution. Finally, performance of the testbed depends on the number of routers and interconnects, thus repeatability is maintained only across the same experiment configuration.

Realistic network conditions. As the ultimate goal of the researcher or developer is to have the protocol deployed, the testbed should offer the ability to subject the protocol under scrutiny to realistic topologies, bandwidth, latency, packet loss and other failure conditions. In addition, the testbed should be able to carry live traffic from Internet users, or those that have opted in to try out the new protocol.

It is not our goal to create a platform to research data plane forwarding performance, since this is highly dependent on the specifics of the forwarding hardware in use. Instead, we target research on control and data plane correctness and functionality. As a result, MX is not designed to carry live, multi-gigabit link bandwidths. Yet, we try to make most efficient use of the available hardware to provide an aggregate bandwidth as high as possible.

Like ModelNet, MX can emulate all bandwidth, latency, packet loss, and link failure conditions. These problems can be modeled as either transient or permanent. Furthermore, MX can emulate router failures, and we plan to add a router NIC failures emulation capability in the future. We also plan to incorporate the ability to replay live traffic and observed faults, which can be useful when subjecting the protocols to the real world routing problems, e.g., [1].

Scalability. The testbed should scale well with increasing number of underlying components (routers, links), as well as increasing number of flows and their volume. Certain protocol features may only manifest themselves under high loads or high failure event rates. Thus, it is important that the testbed can support such scenarios.

We achieve good scalability by using the following approaches:

- We minimize the use of locks and use lock-less data structures to the fullest extent possible.
- We eliminate important overheads in packet processing by running all data plane modification tasks in the same kernel.
- We try to make an effective use of the L2 (and in recent architectures, L3) caches. Accessing memory can be time-consuming, especially since packets are spending a considerable time within the system. This time is directly proportional to the bandwidth-delay product.

MX's performance depends on the number of routers per core. There is a lesser degree of dependency on the CPU utilization, while the total memory bandwidth is currently the most important factor [7]. As the number of routers grow, there are more packets in flight. In turn, this increases the total amount of data going through the memory subsystem, which can then become the bottleneck. More recent work [6] demonstrates that modern architectures do not suffer from this bottleneck, allowing for even higher scalability.

MX currently does not scale beyond one machine. However, it is built on ModelNet, which supports multiple forwarders, thus we believe that with certain engineering effort MX can span over multiple machines. Ethan *et al.* [8] point out that the performance penalty caused by increased traffic among forwarders can be reduced by clever partitioning of the topology. Furthermore, using an approach like time-dilation [9] can offer emulation of larger or high-throughput topologies.

Isolation Being able to isolate one experiment from another is a prerequisite for achieving repeatability. Under heavy load from multiple users, the testbed should not suffer the tragedy of commons, in which all users are experiencing poor performance. Instead, we believe that it is necessary to offer some form of resource allocation and isolation that will provide guaranteed performance to the fullest extent possible.

Our approach for achieving isolation is to have a single-user testbed. As MX requires only a few physical machines, this is a realistic design choice. MX has fast deployment times, enabling a couple of researchers to share a single forwarder using time multiplexing.

Accuracy Often, striving for cost-effectiveness can result in over utilization of resources or unforeseen interference among the components. It is nevertheless important to guarantee accuracy while running high-throughput, complex scenarios. Doing so helps to ensure the repeatability of the experiments. Most importantly, it makes the results obtained on the testbed trusted and repeatable in live deployments.

MX maintains accuracy by implementing requirements for isolation and scalability.

Cost-effectiveness. A high-performance testbed can almost certainly be built by employing extensive hardware resources. However, we believe that a testbed should have a low barrier to entry, to increase the number of researchers that can use it. Thus, the testbed should make efficient use of low-cost, commodity components such as off-the-shelf servers and networking switches and cards.

MX tries to maximize the utilization of the hardware resources, and therefore represents a cost-effective testbed. It leverages the ubiquity of multiple cores and commodity servers.

3. EVALUATION

In our evaluation, we address the following questions: **1)** How well does MX scale compared to fully virtualized environments? **2)** How do the L2 cache and router placement affect the testbed performance?

3.1 Experimental Setup

In order to answer these two questions we conducted three experiments. Our hardware platform is an Intel SR1560 Series rack 1U server with 2 Intel Quad-Core Xeon X5472 processors running at 3 GHz (Figure 3). Each CPU is equipped with 12 MB of L2 cache (2 x 6MB). The machine has 8 GB of 800 MHz RAM that is accessible over the 1600 MHz Front Side Bus. We used FreeBSD-RELEASE-p3 7.1 for the MX related experiments, running Click 1.6.0 with our patches to make it work under FreeBSD.

Our point of comparison is a setup in which multiple Click instances run in separate kernels, on top of Xen. This setup (called Xen/Click from now on) corresponds to the existing cost-effective approach for networking testbeds (the more expensive alternative being to run one Click instance per physical machine). For these experiments we use Debian Linux with the 2.6.26-1-xen-amd64 kernel, whereas Click was the same 1.6.0 version as in the MX setup. To replicate MX's ability to emulate link conditions, we interconnect Click instances via virtual links. The NICs sharing the same virtual link were connected to the same bridge. The link parameters were created using `tc` [3]. With `tc` we were able to set a virtual link's bandwidth and latency, which we later verified with `ping` and `netperf`.

In all of our experiments we used the string topology in which every router (except the first and the last) has one predecessor and one successor. The first router in the row generates packets using the Click's InfiniteSource element that are addressed for the unused NIC of the last router. Each router runs a standard IP router Click configuration and touches each packet. At the last router we discard all packets, while keeping track when we received the first and the last packet. This procedure enables us to measure the throughput of the configuration.

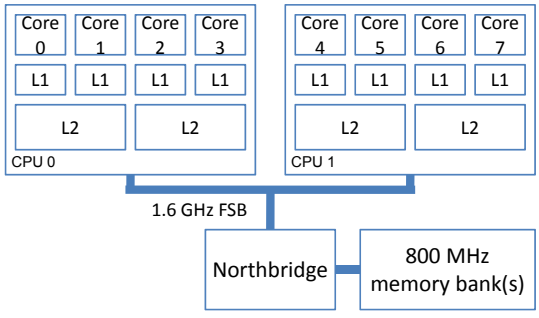

Figure 3: CPU and memory architecture of the Intel SR1560 server.

3.2 Scalability

In the first experiment we compare the scalability of MX and Xen/Click by measuring the performance of forwarding small packets (74 bytes in size). We vary the number of routers involved, as well as the router placement (Figure 5 shows the placement details). Each box in Figure 5 represents one core in the system, while each number represents one router. Arrows point in the way packets traverse the system. Figure 4 shows the throughput (in thousands of packets per second) of each configuration. As the number of routers increases, MX performance becomes several orders of magnitude better than Xen/Click. As configuration D and E on Figure 4 shows, Xen/Click based solution is able to pass only 17.71 and 29.84 packets per second, respectively. In these configurations, there were 4 routers running on the same core; the biggest impact on the performance was switching between VMs. As each Click router in MX runs as a kernel thread (and packets are not even crossing the kernel boundaries) MX scales significantly better.

Xen/Click performs better than MX only in the configuration with a single Click router, which prompted us to inspect possible causes. It turns out that Click is more optimized and up-to-date for Linux. Some parts of the FreeBSD-related code were not updated since FreeBSD 4.1. Also, the scheduler and task related sections were not as up-to-date as their Linux counterparts.

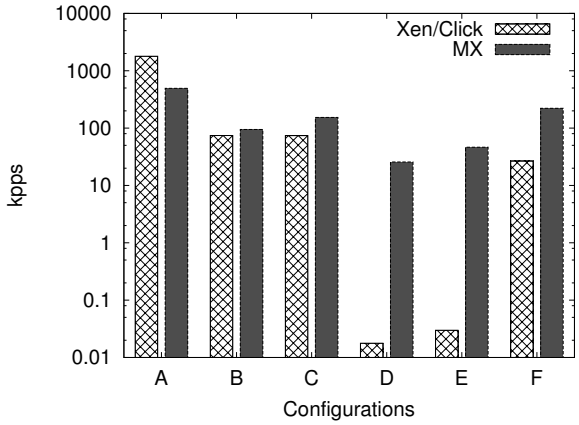

Figure 4: Comparison of scalability of Xen/Click and MX under different configurations (y-axis is in log-scale).

3.3 Router Placement

Figure 3 shows the memory architecture of the server we use for our experiments. Each core has its own small L1 cache, and L2 caches are shared by pairs of cores. Each CPU is connected to the memory banks though the northbridge via one Front-side bus. The major obstacle in achieving high throughputs in such environment is the hierarchical structure of memory, and the limited throughput of the memory bus.

Once a packet reaches the NIC, the driver transfers it to main memory using DMA. Then the packet is copied to the L2 cache of the core which processes it. If the next core in line to process this packet does not share the L2 cache with the previous core, the modified portion of the packet will be transferred back to main memory and subsequently the whole packet will be copied to the L2 cache of the following core.

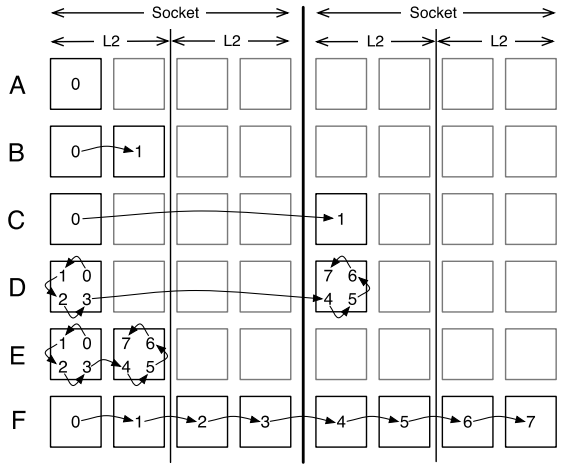

Figure 5: Configurations used in scaling experiments (Figure 4).

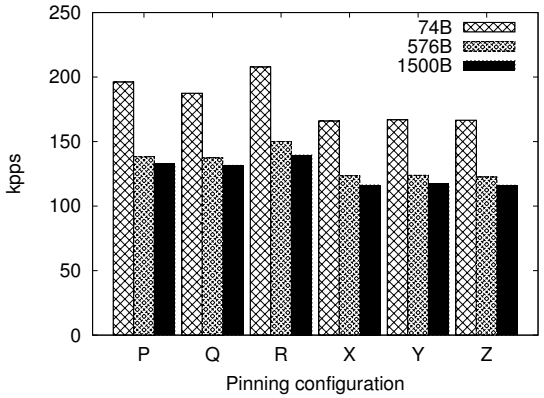

Figure 6: Throughput of the same MX configuration with 4 routers, using different router-to-core pinnings.

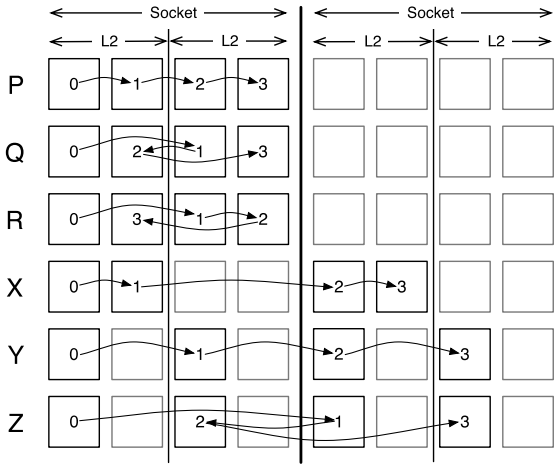

Figure 7: Description of different pinnings used in router allocation experiments (Figure 6).

If these actions repeat for each hop in our virtual topology, significant memory bandwidth is lost on these transfers. Moreover, the whole system slows down, because new packets can not be transferred. Thus, it is important to keep the packets in L2 cache as much as possible.

Figure 6 shows how the MX throughput changes as we vary the placement of the routers (depicted in Figure 7). In this experiment, the emulated links between routers are set to have infinite bandwidth. For each configuration, the throughput is shown for 74, 576, and 1500-byte packets. As expected, larger packets increase the throughput that we can achieve, although the number of packets drops. We suspect this is due to the routers spending more time with larger packets.

At the extreme, the configurations on the right-hand side of the bar chart exercise more packet transitions from one CPU socket to another, putting stress on the memory system. As seen in Configurations X, Y, and Z, as soon as we start shipping packets between different CPU sockets, MX forwarding performance starts to decrease. Configuration P, Q, and R see a smaller drop in performance, although there is the same number of transitions as in Configurations X, Y, and Z, due to Intel's Cache snooping filter [2], which saves memory bandwidth if the data is in a nearby cache.

Figure 6 also supports a hypothesis that the increase in the number of different L2 caches touched by a packet results in lower performance than the case involving a single L2 cache.

4. RELATED WORK

4.1 Routing and Forwarding

MX is built using Click [14] and ModelNet [15] (for the forwarding plane), and XORP [10] (for the routing control plane). The Click [14] modular router is an open-source framework for building routers. Click was designed to be efficient and fast, while maintaining extensibility. Special care was taken to make Click work efficiently on multi-core systems. XORP [10] is an extensible open-source routing platform which offers implementations of many routing protocols.

The work by Egi *et al.* [7] shows that the forwarding performance of modern software routers is rather good. It further shows that L2 caches and unified memory architectures represent a bottleneck even for a single router running on commodity hardware. Along similar lines, Dobrescu *et al.* [6] propose to use a cluster of machines to achieve high throughput of a single IP router. Instead of targeting raw forwarding bandwidth, MX emulates an entire network topology on a single multi-core machine.

4.2 Testbeds

Emulation testbeds provide users with great control over the host and network environments and offer easy reproducibility, at the expense of live network conditions. Wide-area testbeds provide real network conditions with less repeatability and control over the experiment. Emulab [16]

was one of the first large-scale network emulators. It is a shared infrastructure, with many machines, routers and interconnects. To provide isolation of users, at most one experiment runs at any time. Users need to load their own virtual machines before the experiment. This contrasts with quick deployment in MX, where users just need to load the topology and router configurations.

Work by Hibler *et.al* [11] improves Emulab's scalability by leveraging FreeBSD's jails [12] to virtualize all resources. Jails offer the minimum level of virtualization that provides transparency to applications. Since jails cannot run kernel modules, they cannot run modified routing plane software (e.g., Click) in kernel mode (which is required for high performance). Further, because MX has fewer overheads we believe it requires fewer physical machines to emulate the same extensible router topology.

ModelNet [15] is a large-scale network emulator, that supports hundreds of thousands of links. It offers accurate emulation of link losses, delays and bandwidth. Unfortunately, it does not support any data plane packet modification (e.g, cannot run traceroute), and all packet routes are computed and set before the experiment. The only known extensions to ModelNet that enable data plane modifications were XCP-related and were not made public [17].

VINI [4] is a virtual network infrastructure that allows network researchers to evaluate their protocols and services in a realistic environment. VINI uses XORP and Click, over a logical network topology; OpenVPN is used to connect the nodes in a virtual network. Unlike MX, VINI is a shared infrastructure running across many nodes, thus sacrificing some controllability. PL-VINI, a version of VINI running over PlanetLab, suffers from low forwarding speeds as it runs programmable routers in user mode.

A new implementation of VINI called Trellis [5] improves the performance and capabilities of PL-VINI by moving the virtual networking into the Linux kernel, thus enabling faster basic packet forwarding and traffic shaping via standard Linux tools. Trellis leverages container-based virtualization (like VServer and NetNS) to achieve isolation and improve performance. In this sense Trellis is similar to MX. However, we are reducing the time a packet has to spend in the networking stack, thus improving the throughput further. In our approach, forwarding is done in the kernel, while in the case of Trellis forwarding with data plane modification is done in the user mode.

Keller *et al.* [13] offers an approach for running multiple virtual routers on a single piece of hardware using source code virtualization. Instead of compartmentalizing the router resources, the configurations are merged into the configuration of a single, large router. As in our approach, each routers' configuration runs in a separate kernel thread. In contrast with our goal of emulating an entire topology within a single machine, their work improves the performance when a physical machine is shared by different router instances in multiple virtual networks.

5. CONCLUSIONS

We presented MX – an inexpensive large-scale networking testbed. As previous work shows, forwarding performance of modern software routers is rather good. Following these results, we show that a modern x86 server can take a role of a large-scale network testbed, emulating link characteristics as well as enabling data plane modification at the endpoints of these links. By isolating each Click router in a separate kernel thread, we allow one physical machine to behave as multiple routers. Further, we added Modelnet to connect the routers with virtual links to emulate a full topology. This approach in building networking testbeds can allow researchers to quickly and inexpensively test their protocols.

Acknowledgments

This project is supported in part by a grant from the Hasler foundation (grant 2103). Simon Schubert is supported in part by a Microsoft Research PhD Scholarship.

6. REFERENCES

[1] Czech provider causing mayhem. `http://www.renesys.com/blog/2009/02/longer-is-not-better.shtml`.
[2] Intel 5400 chipset memory controller hub (mch). `http://www.intel.com/Assets/PDF/datasheet/318610.pdf`.
[3] Linux advanced routing & traffic control. `http://lartc.org/`.
[4] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford. In VINI Veritas: Realistic and Controlled Network Experimentation. In *SIGCOMM*, 2006.
[5] S. Bhatia, M. Motiwala, W. Muhlbauer, V. Valancius, A. Bavier, N. Feamster, L. Peterson, and J. Rexford. Hosting Virtual Networks on Commodity Hardware. Technical Report GT-CS-07-10, Georgia Tech Computer Science, Jan 2008.
[6] M. Dobrescu, N. Egi, K. Argyraki, B.-g. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. RouteBricks: Exploiting Parallelism to Scale Software Routers. In *SOSP*, Oct 2009.
[7] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, and F. Huici. Towards High Performance Virtual Routers on Commodity Hardware. In *CoNEXT*, Dec 2008.
[8] K. Y. Ethan, E. Eade, J. Degesys, D. Becker, J. Chase, and A. Vahdat. Toward Scaling Network Emulation using Topology Partitioning. In *MASCOTS*, Oct 2003.
[9] D. Gupta, K. Yocum, M. McNett, A. C. Snoeren, A. Vahdat, and G. M. Voelker. To infinity and beyond: time warped network emulation. In *SOSP*, Oct 2005.
[10] M. Handley, E. Kohler, A. Ghosh, O. Hodson, and P. Radoslavov. Designing Extensible IP Router Software. In *NSDI*, May 2005.
[11] M. Hibler, R. Ricci, L. Stoller, and J. Duerig. Large-scale virtualization in the emulab network testbed. *USENIX ATC*, Jan 2008.
[12] P.-H. Kamp and R. N. M. Watson. Jails: Confining the Omnipotent Root. In *SANE*, May 2000.
[13] E. Keller and E. Green. Virtualizing the Data Plane through Source Code Merging. In *PRESTO*, Aug 2008.
[14] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click Modular Router. *ACM Trans. Comput. Syst.*, 18(3), 2000.
[15] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker. Scalability and Accuracy in a Large-Scale Network Emulator. In *OSDI*, Dec 2002.
[16] B. White, J. Lepreau, L. Stoller, and R. Ricci. An Integrated Experimental Environment for Distributed Systems and Networks. *ACM SIGOPS Operating Systems Review*, Jan 2002.
[17] K. Yocum and J. Chase. Explicit Rate Control for Anypoint Communication. Technical Report CS-2004-05, Duke University, July 2004.