Towards Hardware Accelerated Software Routers

Nadi Sarrar Anja Feldmann Steve Uhlig
TU Berlin / Deutsche Telekom Laboratories, Germany
{nadi,anja,steve}@net.t-labs.tu-berlin.de

Rob Sherwood Xin Huang
Deutsche Telekom Inc. R&D Lab, USA
{r.sherwood,xin.huang}@telekom.com

ABSTRACT

We propose an alternative router design to fill the gap between PC-based open source software routers and commercial high-end routers. Our hardware accelerated software router uses commodity PC hardware running an open source software router for the control path, and couples it with programmable switching hardware by delegating most of the packet forwarding to the switch. We describe a prototype implementation and show that it is capable of handling the traffic requirements of an existing carrier aggregation network.

1. INTRODUCTION

Open source routing software [3] is already deployed within enterprise and ISP networks for specialized tasks. Its quality is approaching what is commonly referred to as "carrier grade". However, PC based routers with open source routing software do not offer the throughput, port density, and/or reliability needed for ISPs' purposes. On the other hand, we observe that Ethernet switches often provide layer-3 functionalities and contain components, e.g., TCAMs, for performing longest prefix matching at line rate and offer multiple terabit throughput.

In this paper, we leverage the decoupling of the traditional tasks of a router, i.e., maintaining up-to-date routing information and forwarding packets. We propose an alternative router design that couples a commodity PC running an open source software router with lower-cost programmable switching hardware. Such switches usually have very limited FIB memory, not enough to keep a complete Internet routing table of more than 300*k* entries. We are using the switch as a flexible forwarding engine for high-performance forwarding for most of the traffic. The PC acts as a route controller and furthermore handles the traffic that is chosen not to be forwarded by the switch. Our prototype is based on an OpenFlow-enabled switch [2], and a Linux-PC running the open source routing suite Quagga [3]. We have implemented an OpenFlow controller, **RouteVisor**, that serves as the glue logic combining Quagga and the switch.

We believe that our approach has the potential to bring the flexibility of software routers into previously unexplored contexts, such as data centers and aggregation networks.

ACM CoNEXT Student Workshop, November 30, Philadelphia, USA.

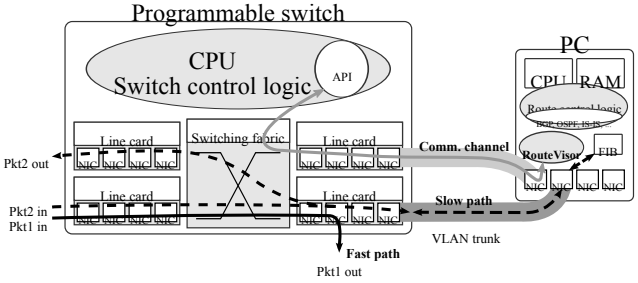


Figure 1: Our alternative router design with its slow path (dashed black arrow), fast path (solid black arrow), and communication channel (grey arrow).

2. DESIGN / IMPLEMENTATION

Figure 1 illustrates the components that form our router design, along with their interactions. The programmable switch serves as our fast path, the PC as our route controller. Our architecture relies on a communication channel and alternative forwarding path (slow path) between the switch and the PC, maintained by our controller **RouteVisor**.

2.1 Programmable switch

Crucial for our router design is the availability of an advanced switch control interface, such as Openflow [2], for modifying FIB contents as well as for retrieving traffic statistics. For our prototype, we use the HP ProCurve 5406zl which supports up to 2000 wildcarded FIB entries which we use as a FIB cache. It takes roughly 90ms to obtain statistics for 1000 FIB entries, and the modification of a single FIB entry is completed in 3ms on average.

2.2 RouteVisor

The main tasks of **RouteVisor** include 1) allowing Quagga to operate on physical switch ports as if they were local interfaces, 2) monitoring routing table updates by Quagga, 3) maintaining traffic statistics, and 4) populating the FIB of the switch with the most popular destination prefixes. Although we rely on Quagga for our prototype, any other Linux software router should work as a plug-in re-

placement, since no source code modifications to Quagga are necessary.

To meet 1), **RouteVisor** establishes a VLAN trunk between switch and PC, with unique VLAN identifiers per switch port, providing visibility of all physical switch interfaces to Quagga. The VLAN interfaces are configured to see only routing protocol messages, ARP as needed, and slow path traffic. Any router adjacent to the switch can then establish a BGP, OSPF, or IS-IS session with Quagga. The Linux networking stack forwards packets across the VLAN interfaces. To the outside world, the switch-PC combination acts as if it was a conventional IP router.

For 2), **RouteVisor** monitors changes to the PC's routing table by Quagga via the Linux netlink subsystem. In addition, to achieve 3), **RouteVisor** maintains packet counters per routing table entry for both slow path, via Linux forwarding table counters, as well as fast path, through OpenFlow statistics queries.

The information gathered in 2) and 3) is evaluated by our prefix selection strategy **FIBpredict**. With respect to 4), **RouteVisor** periodically calls **FIBpredict** to compose a new selection of prefixes which are expected to contribute most of the upcoming traffic, and updates the switch FIB accordingly. Incoming BGP updates that change forwarding decisions are immediately propagated to the switch.

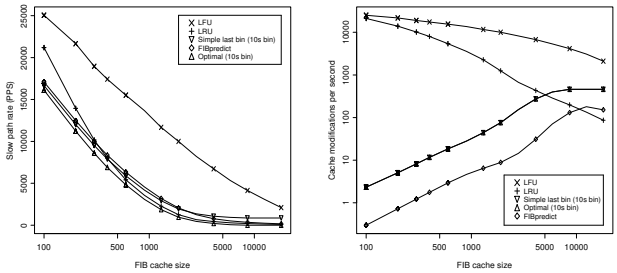
2.3 FIBpredict

The design criteria for **FIBpredict** are to keep most traffic on the fast path while minimizing FIB churn and making it hard for an attacker to target the slow path. To keep most of the traffic on the fast path we rely on the Zipf-like property of destination popularity in Internet traffic and use past traffic statistics to predict upcoming popular prefixes. To keep churn low we rely on long term trends, and to ensure flexibility and short reaction times, e.g., in case of traffic changes or DoS attacks, we also include recently popular prefixes.

2.4 Forwarding

As we are constrained by limited FIB memory on the switch, we introduce a fast path and a slow path for packet forwarding, as depicted on Figure 1. If a packet (Pkt1) matches a prefix as present in the switch FIB, the switch can forward the packet on its own, we call this our fast path. The OpenFlow switch takes care of rewriting the MAC addresses, and the upcoming release of OpenFlow 1.1 introduces support for decrementing IP TTL.

If the destination of a packet (Pkt2) is unknown to the switch FIB, it is directed towards the PC via the VLAN trunk. The PC learns about the packet's ingress port by inspecting the VLAN identifier. Then, Linux determines the appropriate egress interface based on its local routing table as maintained by Quagga. Finally, after rewriting the layer-2 header and decrementing the IP TTL value, the packet is sent back to the switch where it is then forwarded to the appropriate network interface based on the VLAN identifier.



(a) Slow path rate (log-linear). (b) FIB cache churn (log-log).

Figure 2: FIBpredict performance.

3. PERFORMANCE

In this section we show that our **FIBpredict** strategy performs sufficiently well for handling traffic demands of a carrier aggregation network. We base our study on an anonymized 48 hours packet-level trace of 20.000 residential DSL lines collected in 2009 at an aggregation point within a large European ISP [1]. We focus our study on the required slow path capacity and the number of FIB modifications needed, both as a function of FIB cache size.

Figure 2(a) shows the impact of FIB cache size on the resulting slow path rate for different prefix selection strategies. The optimal one updates the FIB every 10s and has perfect knowledge. The simple strategy selects the most popular prefixes as seen during the last bin. **FIBpredict** is our strategy which relies on both short-term and long-term trends. For a reasonably sized FIB, all three strategies keep the slow path rate low. Figure 2(b) however shows that **FIBpredict** outperforms both the optimal and the simple last bin strategies substantially in terms of FIB modifications per second.

We also compare against LRU and LFU, whose applicability is questionable since for each cache miss, a FIB modification is needed before the packet can be forwarded.

4. SUMMARY AND FUTURE WORK

In this work we propose a hardware accelerated software router design and present a prototype based on Quagga, OpenFlow, and our controller **RouteVisor**. In future work we will be investigating its applicability to different deployment scenarios, e.g., data centers. Furthermore, we plan to enhance **RouteVisor** such that it actively detects malicious activity and provides countermeasures against attacks.

5. REFERENCES

[1] MAIER, G., FELDMANN, A., PAXSON, V., AND ALLMAN, M. On dominant characteristics of residential broadband internet traffic. In *ACM IMC* (2009).

[2] MCKEOWN, N., ANDERSON, T., BALAKRISHNAN, H., PARULKAR, G., PETERSON, L., REXFORD, J., SHENKER, S., AND TURNER, J. OpenFlow: enabling innovation in campus networks. *ACM CCR* (2008).

[3] Quagga Routing Suite. `http://www.quagga.net`.