

Deutsche Telekom Laboratories
An-Institut der Technischen Universität Berlin

FIBIUM: Towards Hardware Accelerated Software Routers

Nadi Sarrar, Anja Feldmann, Steve Uhlig, Rob Sherwood,
Xin Huang

Technical Report No 9
November 2010

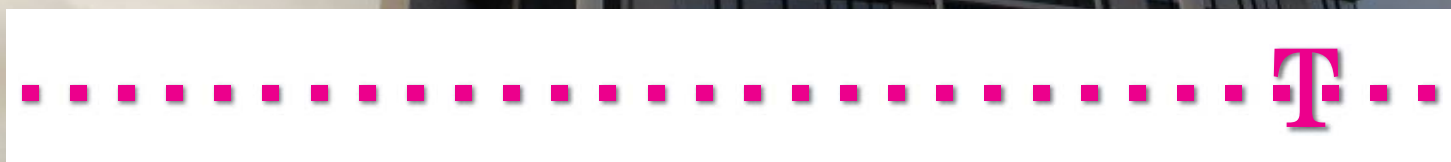


Table of contents

1	Introduction	3
2	Router/Switch architectures	4
2.1	Router architectures	4
2.2	Routing protocols	4
2.3	Switch architectures.....	4
2.4	ISP networks and traffic	5
3	Design	5
3.1	RouteVisor	5
3.2	Forwarding	5
3.3	Programmable switch	6
3.4	Summary	6
4	Prototype Evaluation	6
4.1	Implementation	6
4.2	Performance characteristics	7
5	Feasibility	7
5.1	Validation datasets	7
5.2	Optimal caching	8
5.3	FIB cache churn	8
5.4	Scalability	9
5.5	Impact of misprediction	9
6	FIB management strategies.....	10
6.1	Traditional reactive caching	10
6.2	Traffic workload analysis for FIBpredict.....	10
6.3	Multiple time scales FIBpredict.....	11
6.4	Attacks against FIBpredict	12
7	Related work	12
8	Summary and future work	13
9	Table of figures	14
10	References	15

Abstract

In this paper we propose a hardware accelerated software router, called **FIBIUM**, that (a) uses commodity PC hardware running an open source software router for the control path and (b) couples it with programmable switching hardware by delegating most of the packet forwarding to the switch.

We propose an alternative router design to fill the gap between pure software routers that suffer from low forwarding performance and commercial routers. The key component of **FIBIUM** is the intelligence applied to the choice of which forwarding entries to keep on the switch hardware to maximize the forwarding performance of the software router.

We describe a prototype implementation and show that it is in principle capable of handling full IP routing tables and the traffic requirements of a carrier aggregation network.

1 Introduction

Open source routing software [1, 10, 17, 19, 37] is already deployed within enterprise and ISP networks for specialized tasks. Its quality is getting closer to what is commonly referred to as “carrier grade”. However, PC based routers with open source routing software do not yet offer the throughput needed for ISPs’ purposes [9, 11, 25].

Still, recent advances in both hardware and software capabilities motivate us to revisit the question of what may be a good architecture for a hardware accelerated software router. Current Ethernet switches can often be used as layer-3 switches. Such layer-3 switches have limits as they are not designed to replace backbone routers. Nevertheless, current switches, e.g., from vendors such as HP, NEC, Cisco, 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: maintaining up-to-date routing information and forwarding packets. We propose an alternative hardware accelerated software router, called **FIBIUM**, that (a) uses commodity PCs running an open source software router for the control path [9, 11, 25] and (b) couples it with lower cost programmable switching hardware by delegating packet forwarding to the switch, see Figure 1(b).

We are using the switch as a flexible forwarding engine for high-performance forwarding. The commodity PC serves both as a routing controller as well as for handling the traffic that was chosen not to be forwarded by the switch. With our approach, we have a wealth of new algorithmic opportunities to design smart forwarding entry selection strategies that maximize the number of packets handled by the switch while minimizing the number of forwarding entry updates on the switch. Moreover, such an algorithm can be designed for example to adapt arbitrarily fast to changes in the routing tables, or to changes in the traffic pattern. Additionally, the algorithm can be robust and can limit attacks against the route

controller. By leveraging the popularity of IP prefixes that follows a Zipf law [6, 12, 46, 52], we show that it is possible to design a prefix-based forwarding entry selection strategy that exhibits very close to optimal performance and outperforms traditional caching strategies, e.g., LRU and LFU.

The advantage of our proposal is that we (a) separate the development cycle for hardware based forwarding and control software, (b) gain flexibility as routers are increasingly expected to do more than route packets, (c) have a hardware accelerated software based alternative to traditional routers such as those by Juniper and Cisco, especially as switches are becoming a commodity. We do not claim that our approach will lead to alternatives comparable to current high-end switches and routers available on the market, neither in terms of forwarding performance nor features supported. However, we believe that our approach has the potential to bring the flexibility of software routers in previously unexplored contexts, e.g., data centers and aggregation networks.

The remainder of this paper is structured as follows. We first provide some background information about switches and routers (Section 2). Next, we explain our design choices (Section 3). We describe our prototype implementation and some related performance aspects in Section 4. We evaluate the feasibility of our approach in the context of a carrier aggregation network in Section 5 and compare its performance to known caching techniques in Section 6. Related work is covered in Section 7.

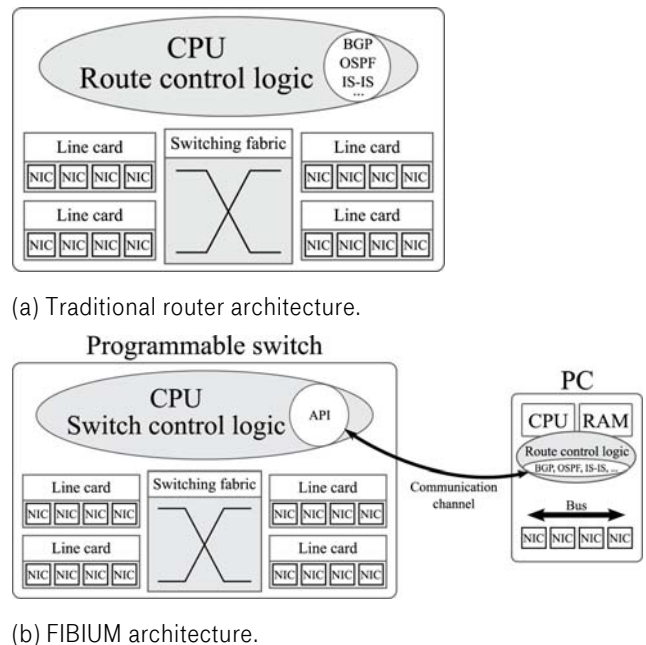


Figure 1: Architectural overview of current IP routers and FIBIUM.

2 Router/Switch architectures

In this section we briefly review router and switch architectures to highlight some of the differences. Moreover, we provide a brief review of routing protocols and traffic demands. The Internet is divided into a collection of autonomous systems (ASs). Routing through the Internet depends on protocols for routing between ASs, e.g., BGP [16], and protocols for routing within individual ASs, e.g., OSPF, IS-IS [32]. Moreover, multiple kinds of technologies are used to run IP networks, even within a single AS. This set of technologies includes packet-over-SONET, FDDI, ATM, and Ethernet. However, Ethernet is becoming the technology of choice as it offers a low cost alternative [15].

2.1 Router architectures

Each router terminates a mixture of access, peering, and backbone links. Routers perform several tasks among which: (1) computing the routing table for each routing protocol, (2) computing a forwarding table based on the routing tables, (3) forwarding packets, (4) managing network interfaces using different technologies, (5) limit access to network resources through access control lists (ACLs). Tasks 1 and 2 are done in software on the route processor, see Figure 1(a) and Section 2.2 for more details. Tasks 3, 4 and 5 are typically done in hardware on the line cards in high-end routers.

The first generation of routers were designed very similar to today's PCs. A data bus or shared memory was used to interconnect multiple network interface cards (NICs) and most operations on the packets were carried out by the CPU. However, the required throughput between NICs soon exceeded the available capacity. Therefore, the next generation of routers relied on line cards which augmented the capabilities of NICs by increasing their processing power and memory size. The memory on the line cards was used to store forwarding information. This architecture is closer to Figure 1(a), but still relies on a data bus or shared memory to move packets across line cards.

In the early 90's, route caching was seen as a possible solution to speed up forwarding [4]. However, this approach was soon abandoned due to the complexity of cache management. With the availability of fast and cheap memory, the full routing table could be stored on the line cards, making caching unnecessary.

Replacing the data bus with a switching fabric enabled terabit routers, see Figure 1(a). The switching fabric allows line cards to exchange packets at very high speeds, without involving the route control logic. Almost all high-end commercial routers, including those of Cisco and Juniper, rely on such an architecture [43]. The switching fabric is the fast path of a router. Some CPU time is used for the slow path which only handles non-standard packets and control packets.

Line cards on IP routers are therefore optimized to minimize the number of packets that have to be forwarded through the slow

path. Given current line rates, high-end line cards have to utilize fast memory flavors, e.g., SRAM, RLDRAM, or TCAM, as well as specialized ASICs [48], thus increasing their price tag.

2.2 Routing protocols

Today's routers have to support many routing protocols including BGP, IS-IS, OSPF, RIP. Most routing protocols maintain a routing table that contains the best route to each neighbor, called routing information base (RIB), based on the information it receives from its neighbors. Information from these RIBs is combined to build a forwarding table, the forwarding information base (FIB). The FIB lists for each destination prefix the output port to which the packet has to be sent and is maintained by the route controller.

Today, a full BGP RIB typically contains more than 300,000 entries, as BGP is used to ensure Internet wide reachability. OSPF/IS-IS RIBs typically contain only entries for the routers within a single AS. As such their RIBs are smaller. Nevertheless, the resulting FIB contains several hundred thousand entries. This number is expected to further increase as the Internet is still growing and IPv6 is starting to be used, giving rise to two different kinds of concerns [18]: (a) FIB size, as the line cards need to have sufficient memory to maintain a full copy of the FIB; (b) routing dynamics, as updates need to be pushed to the line cards.

Given that BGP prefixes are typically globally visible in the Internet, any change or update as well as any link failure can lead to routing updates. Today, the number of routing updates roughly corresponds to a few updates per second with peak rates of up to a few thousands. However, every update might require the re-computation of the routing table and a modification of some of the FIB entries of each line card. Therefore, it is important to have enough resources for routing to not disrupt forwarding.

2.3 Switch architectures

Current switches and routers are conceptually quite similar: they both forward packets and contain fast switching fabrics. However, they differ in their concepts: switches are layer-2 devices while routers operate on layer-3. This implies that from a software perspective switches are in principle simpler as no support for routing protocols is needed. However, they have to support spanning trees to prevent forwarding loops at layer-2.

While routers support a huge range of technologies, switches only support Ethernet. Ethernet is becoming the technology of choice even for wide-area interconnection due to its price-performance benefits [15].

Contrary to routers, layer-2 switches do not need huge FIBs nor do they need to support longest prefix matching. Their lookup is based on a fixed size identifier, the MAC address. However, switches also have to handle the dynamics of their forwarding tables due to the growth of layer-2 islands and the increased popularity of mobile devices, without mentioning requirements in

FIBIUM:
Towards Hardware Accelerated Software Routers

sub-millisecond spanning tree convergence times, as well as resilience to attacks.

Carrier-grade layer-2 switches offer some layer-3 capabilities, e.g., ACLs, traffic shaping, and even support somerouting. For that, TCAMs are used—a technology also used in router line cards, e.g., for the purpose of ACLs. As such, the capabilities of a layer-3 switch are in principle getting closer to those of routers. However, the current size of a switch TCAM is usually at most a few thousands of entries while high-end router line cards must be capable of performing longest prefix matching on 300,000 entries at line rate [48].

2.4 ISP networks and traffic

ISP networks consist of a collection of IP routers and bidirectional layer-3 links. Customers are typically connected to an edge router. Traffic by such edge routers is then aggregated via an aggregation network sometimes consisting of multiple routers. These are then connected to a backbone router which is part of the core network.

A reoccurring property of Internet traffic at various levels of aggregation is its consistency with Zipf’s law [6, 12, 46, 52]. In particular, the amount of traffic per destination prefix is consistent with a Zipf distribution. This implies that a small fraction of the destination prefixes is responsible for most of the traffic.

However, which prefixes are popular at any given time can and does change [34, 46, 47]. This implies that we have to address the question of how to predict which prefixes are going to be popular and change the FIB entries accordingly.

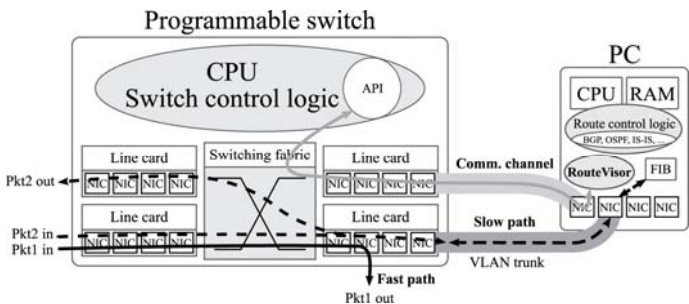


Figure 2: FIBIUM design with its slow path (dashed black arrow), fast path (solid black arrow), and communication channel (grey arrow).

3 Design

In Section 2.1 we highlighted that the main tasks of a router include the computation of the routing and forwarding tables as well as the actual packet forwarding. Software routers are typically run on commodity PC hardware, thus their major drawbacks include (a) limited packet forwarding performance and (b) limited port density. With FIBIUM, we propose a novel way of bringing together a software router with an Ethernet switch (Figure 2).

The switch provides a hardware accelerated forwarding path as well as a higher port density as compared to PCs. This fast path is used as a forwarding engine for most of the traffic. The PC serves not only as a route controller, but it also monitors the traffic and maintains a communication channel with the switch to populate its FIB with the most popular destination prefixes. In addition, it provides a slow path to forward the remaining traffic not handled by the switch. In Sections 5 and 6 we show, that given current hardware and software, our FIBIUM prototype can in principle handle the traffic volume of a carrier aggregation network.

3.1 RouteVisor

Instead of modifying existing open source routing software, we opt for a design in which we can use any software router as a route controller. Our switch controller, **RouteVisor**, creates virtual siblings of the switch ports on the PC, providing visibility of all physical interfaces to the software router. Those sibling interfaces see only slow path traffic not handled by the switch, as well as all traffic necessary for routing protocols to operate. As a consequence, any adjacent router connected to one of the physical switch ports can establish, e.g., a BGP peering session with the route controller. To the outside world, the switch-PC combination acts as if it was a conventional IP router.

In addition, **RouteVisor** needs to decide for a set of prefixes to be installed into the switch FIB. For that, it collects traffic statistics from both the slow path and the fast path. A smart prefix selection strategy implemented within **RouteVisor** then uses these statistics to periodically compose a selection of prefixes which are expected to contribute most of the traffic.

RouteVisor monitors if changes to the FIB on the PC require a modification of the switch FIB. For example, the removal of a prefix in consequence of an incoming BGP withdrawal should immediately be propagated to the switch. Also, **RouteVisor** should be able to detect malicious activity and provide countermeasures against attacks.

3.2 Forwarding

In FIBIUM, most of the traffic is forwarded by the switch. We benefit from the destination lookup performance of linecards present in current switches as well as the high throughput achieved by switching fabrics. Given that we want to rely on commodity switches that have only limited FIB memory, we cannot install all FIB entries that current routers have to keep into the switch.

FIBIUM:

Towards Hardware Accelerated Software Routers

Accordingly, we introduce a fast path and a slow path for packet forwarding, see Figure 2. If a packet, Pkt1 in Figure 2, is forwarded using the fast path, the relevant FIB entry has already been loaded into the switch FIB by **RouteVisor**. In this case the switch can handle the packet forwarding entirely by itself.

If the destination of a packet, Pkt2 in Figure 2, is unknown by the switch FIB, it is directed towards the PC via a dedicated network interface. Upon receiving this packet, the PC can use any forwarding software, e.g., an in-kernel packet forwarding stack, to determine the appropriate output interface on the switch. After rewriting the layer-2 header and decrementing the IP time-to-live field, the packet is sent back to the switch where it is then forwarded via the switching fabric to the appropriate network interface.

As long as we do not change entries on the switch FIB, packets for the same destination are either handled on the slow path or on the fast path, keeping delay variance low and packet reordering unlikely. Upon changing a FIB entry on the switch, some packet reordering is possible. However, this reordering is limited to a single TCP window as round trip times in the Internet are significantly larger than the time it takes to update a FIB entry on the switch, see Section 4.2.

A limitation of this approach is the slow path capacity. The prefix selection strategy of **RouteVisor** must make an effort to keep the probability low of overwhelming the slow path with traffic amounts above its capacity.

3.3 Programmable switch

To realize forwarding and switch control as proposed above, **RouteVisor** relies on an advanced control interface provided by the switch. More specifically, a facility to instantiate switch FIB entries and a way to gather traffic statistics per FIB entry is necessary. OpenFlow enabled switches [29] provide such an API [31].

OpenFlow offers the following commands, which are crucial for our design:

- FLOW_MOD: Add and remove individual FIB entries.
- STATS_REQUEST: Request traffic statistics per FIB entry, including packet and byte counters. Each request can refer to a specific FIB entry or a set of entries including the whole FIB.

To understand the impact of the extra delay imposed by the switch control logic we examine how long it takes before an update to the switch FIB takes effect, see Section 4.2.

OpenFlow switches differentiate between wildcarded and exact-match FIB entries and typically support different amounts of either type of entry. For **FIBIUM**, we rely on wildcarded entries for IP prefixes. Today, for example the HP ProCurve 5406zl supports 2000 wildcarded entries. However, the FIB size of OpenFlow switches is expected to increase as hardware and firmware development advances.

3.4 Summary

Above we outline how to build a flexible router using commodity PC hardware and a programmable switch. **FIBIUM** can improve the performance of PC based software routers significantly. In the remainder of the paper, we present results from our prototype implementation and our traffic analysis to show that our design is scalable: it can in principle handle traffic volumes as seen by a carrier aggregation router.

4 Prototype Evaluation

In this section we briefly describe our prototype implementation. We show that it is capable to interoperate with Juniper and Cisco routers in a multi-protocol setup. Moreover, we use it to derive some performance characteristics regarding FIB update delays and rates. The latter are necessary for our design of smart FIB update strategies.

4.1 Implementation

Our prototype is built around an OpenFlow switch and a PC running the Quagga [37] open source router. Figure 2 illustrates our prototype.

RouteVisor and Quagga run on a single PC, a Sun Fire X4150 running Linux, which also serves as the slow path. **RouteVisor** is able to control OpenFlow capable switches, such as HP ProCurve 5406zl, NEC IP8800/S3640, Quanta LB4G, or the NetFPGA-based OpenFlow switch implementation [33]. Although our current prototype is based on Quagga, any other software router, e.g., XORP [17], can be used as a drop-in replacement without source code modifications.

To realize the virtual sibling ports (Section 3) which are necessary to support Quagga's operation and the slow path, **RouteVisor** automatically configures a VLAN trunk as shown on Figure 2. This VLAN trunk is used to multiplex routing protocol messages as well as slow path traffic from the PC to the switch and vice versa. Quagga can operate on top of Linux VLAN interfaces transparently, as the Linux kernel takes care of adding and stripping off VLAN tags as packets pass through.

We explore how much traffic our slow path can handle by pushing as much traffic as possible for a given packet size. We find, that with packet sizes of 500, 1000, and 1500 bytes we can fill the link capacity of 1 Gbps almost entirely. The achieved packet rate per second is in the order of 80k for 1500 bytes packets to 530k for 60 bytes packets.

Our slow path forwarding engine relies on the FIB of the Linux operating system populated by Quagga. Whenever Quagga modifies the Linux FIB, we may have to also change the FIB on the

FIBIUM:

Towards Hardware Accelerated Software Routers

switch. Therefore, **RouteVisor** passively monitors such changes by registering with the Linux routing table netlink subsystem.

To verify that our **FIBIUM** prototype is compatible to commercial hardware routers, we integrate it into a network consisting of a Juniper router J4350, a Cisco router C2691, and a software router running on a Linux PC. We enable a mixture of routing protocols including OSPF and BGP, and re-inject BGP messages recorded by any of the publicly available BGP monitors, e.g., RIS [39] or Route Views [40], using BGPpreplay [27].

Low FIB update times are crucial for our design. We first examine how long it takes for a routing update to take effect on the slow path, i.e., the PC forwarding table. For this purpose, we send individual BGP updates using BGPpreplay to the Juniper router, which has a direct 1 Gbps Ethernet link to the PC running the software router. We measure the time between when the BGP update is received at the software router (using libpcap) and the time the corresponding route is installed in the Linux FIB. We find that the average is 76 ms with a minimum of 65 ms and a maximum of 100 ms. Similar BGP update delays have been measured on high-end routers [14]. Including the delay for installing a single route into the switch FIB increases the average delay by 1 ms (see below for details). Processing a burst of n BGP updates takes in the order of a minute per 100,000 BGP routes which is again consistent with previous results [3, 14].

4.2 Performance characteristics

For any implementation of **FIBIUM**, the performance of the communication channel has to be well understood. We are now examining implicit performance limitations due to the actual hardware and software components that we have chosen for our prototype. These results are the basis for our scalability and feasibility analysis.

FIB update delay and rate. To understand the restrictions on FIB churn and its delay, we check how long it takes to update FIB entries on a HP ProCurve 5406zl using OpenFlow FLOW_MOD commands. We find a strict linear relationship between the number of modified FIB entries and the time for processing the OpenFlow messages and the change to take effect. It takes between 0.8 ms and 1 ms to modify a single FIB entry. Our results are consistent with numbers reported by router vendors [44]. Given this minimal delay we conclude that the switch controller can easily handle at least a thousand FIB updates per second. Moreover, the bandwidth required to send the FIB updates is negligible.

Traffic statistics. Prefix selection strategies have to rely on traffic statistics from both the slow path as well as the fastpath. The slow path statistics can be gathered via system calls and thus impose no relevant performance overhead. The fast path statistics can be gathered via SNMP MIBs which include per interface traffic summaries. But, their granularity is typically limited. However, most

switches, including OpenFlow enabled ones, offer per FIB entry

statistics. To evaluate the granularity at which we can gather these statistics from the switch we repeatedly query the switch using STATS_REQUEST messages to obtain the traffic statistics of all existing FIB entries. By issuing a single command for a different number of entries we find that the delay d to obtain statistics about n entries follows an approximately linear relationship: $d \approx 1.2 \text{ ms} + n \times 0.01 \text{ ms}$. Thus for a FIB size of 1000 entries it takes roughly 11.2 ms to retrieve all stats. This time is negligible compared to the time granularity at which our proposed FIB strategy modifies the FIB. Routers also offer flow-level statistics such as NetFlow [5] or IPFIX [7]. A similar feature can easily be added to our prototype using sFlow [36].

5 Feasibility

In this section, we show that our design is feasible, i.e., that the communication channel and slow path between switch and PC inherent to our design have sufficient capacity to handle the traffic demands of a carrier aggregation network. More precisely, we use packet-level traces to explore how the capacity of the communication channel and of the slow path depend on the FIB size of the switch, i.e., the fast path. In this section, we use our traces (Section 5.1) to derive an optimal baseline (Section 5.2) for the performance of our FIB-based prefix caching approach. We then relax our assumptions and quantify the rate of churn (Section 5.3) in the FIB cache, how our design scales with real traffic workloads (Section 5.4), and the cost of missing the most important FIB entries (Section 5.5).

5.1 Validation datasets

We base our study on anonymized packet-level observations of residential DSL connections collected at an aggregation point within a large European ISP. Our packet-level monitor, using Endace monitoring cards, allows us to observe the traffic of more than 20,000 DSL lines to the Internet. The data anonymization is performed immediately on the secured measurement infrastructure using the Bro NIDS [35].

We use an anonymized 48 hour packet trace collected in August 2009. While we typically do not experience any packet loss, there are several multi-second periods (less than 5 minutes overall per packet trace) with no packets due to OS/file-system interactions. The routing decisions for this kind of dataset are trivial for the direction towards the customers but challenging with regards to prefix variability for the outgoing direction. Accordingly, we distinguish between incoming and outgoing traffic, the latter is referred to as TROUT. Moreover, we sub-sample TROUT based on IP addresses to explore how our approach scales. For this purpose we use a base sub-sample TRBASE of TROUT which contains the traffic of 20,000 random source IP addresses.

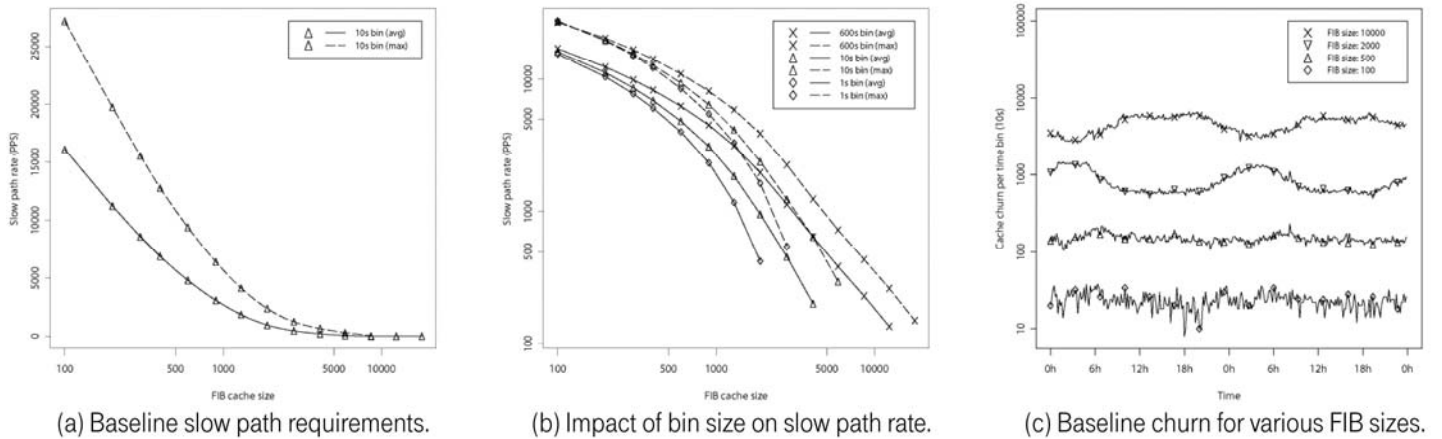


Figure 3: Baseline results for TROUT.

With regards to the application mix of TROUT, Maier et al. [28] found that HTTP, NNTP, BitTorrent, and eDonkey each contribute a significant amount of traffic. Moreover, their total traffic adds up to more than 72% of the overall traffic. Similar protocol distributions have been observed at different times and at other locations in the Internet [26].

5.2 Optimal caching

To evaluate if the capacity of the slow path is sufficient to handle the traffic demands imposed by, e.g., 20,000 DSL lines, we use a simulator that takes the trace as input and estimates the switch FIB hit and miss rates. Put differently, the simulator calculates the number of packets that are handled via FIBIUM's fast path and slow path. This simulator is later used to experiment with various FIB selection strategies. Given that the popularity of prefixes is consistent with a Zipf-like distribution [38, 46, 47] we expect that a significant fraction of the overall traffic can be handled by even a small number of FIB entries.

For the moment, we assume that the FIB cache on the switch can be updated every 10 seconds (Section 4.2 shows this rate is achievable with current hardware). To quantify the lower bound for the FIB rate we further assume that FIB updates are instantaneous and assume perfect future knowledge of the traffic (we relax these assumption below). We call this prefix selection strategy the optimal one. Therefore, we can update the x FIB entries at the start of each 10 second time bin with the x top-ranked prefixes. Figure 3(a) shows the impact of increasing the FIB size on the capacity requirements of the slow path on a log-linear scale. We plot both the average slow path rate over all time bins as well as the maximum. Figure 3(b) includes the same data on a log-log scale.

With 1,000 FIB entries the average required packet forwarding rate on the slow path is less than 2,000, with a maximum of 5,000, only 10% of the fast path rate. When we increase the FIB sizes even further, the requirements for the slow path reduce further. These results are consistent with various previous studies [20–22, 42, 47].

As a thought experiment we now use the above results to scale the traffic up to 10 Gbps, ignoring aggregation gains discussed below (see Section 5.4). In this case a FIB of a few thousand entries is still enough to keep slow path rate to less than a few thousands of packets per second. Previous work has shown that such rates can be easily handled on commodity PC hardware [9] and the communication requirements between the switch and the PC are still well below 1 Gbps.

5.3 FIB cache churn

Because the traffic matrix is constantly changing, the FIB must be continuously updated in order to achieve efficient caching. For this purpose, Figure 3(c) shows the total number of FIB cache modifications required per time bin for 10 second bins on a log scale for different FIB sizes. With only a small number of FIB entries the churn is relatively small. As the number of FIB entries increases the churn increases as well. This is partly due to the fact that a small difference in traffic can cause a larger shift in rank for the lower ranked prefixes than the top ranked prefixes [47].

Also note, that there is an inverse behavior regarding time of day for medium and large FIB sizes. When the FIB is large enough to include prefixes that have limited traffic volume, yet is still too small to include all prefixes (e.g., the FIB size 2,000 in Figure 3(c)), the churn is higher during the times of the day when the total traffic volume is lower. This is explained by the higher stability of the heavy hitters during peak times. As soon as the FIB

FIBIUM:
Towards Hardware Accelerated Software Routers

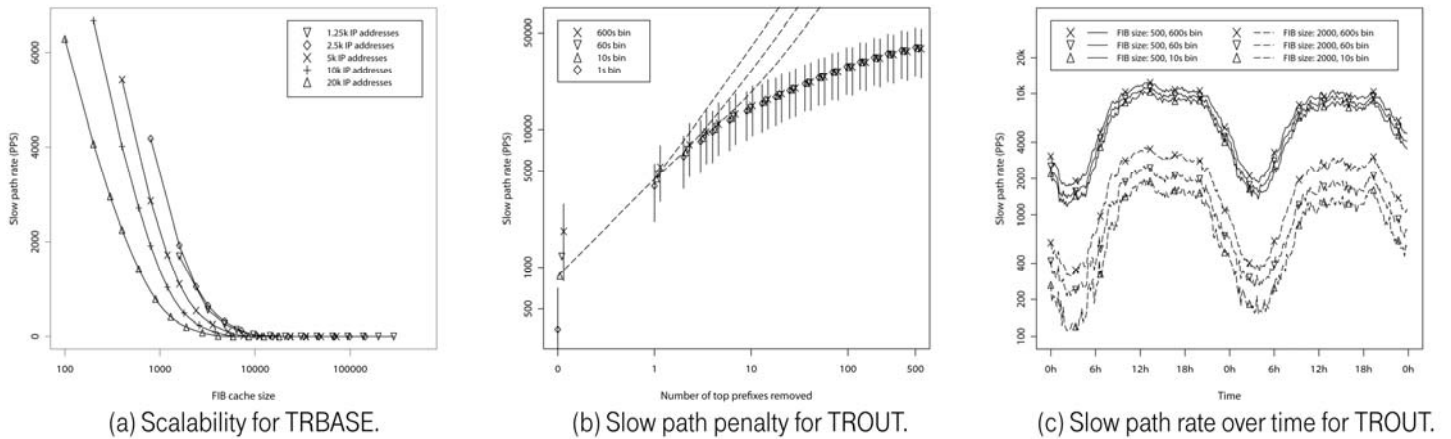


Figure 4: Baseline Scalability and misprediction.

cache size exceeds the total number of prefixes that are seen during any time bin (FIB size of 10,000 in Figure 3(c)), the churn simply follows the time of day pattern. As the total traffic volume increases during peak times, the number of prefixes that have traffic also increases while most of these prefixes have traffic for only very limited periods of time.

5.4 Scalability

So far we presumed that the slow path requirements increase linearly with the traffic volume. However, this extrapolation ignores that it is likely that there is a strong correlation among the top prefixes from multiple different sources. To get an idea of the aggregation gains we sub-sample our trace on a per IP address basis. The starting point for this experiment is the TRBASE—traffic generated by 20,000 randomly selected IP addresses. From these we select 10,000 IP addresses and then repeat the sub-sampling until we have 1,250 IP addresses left. For each one of these data sets we compute the average and maximum number of packets per second that the slow path has to handle as a function of the FIB size.

To be able to compare the results we invert the sampling by multiplying the FIB size as well as the number of packets per second by the sampling factor, see Figure 4(a). We are using the same thought experiment from above but with the sub-samples as the baseline. If we would have no aggregation gain all curves should visually overlap. However, this is not the case. The rescaled results from the sub-samples are shifted to the right. This means that on the basis of the sub-sample we would have predicted much higher resource demands than are actually necessary. Therefore, we conclude that there are aggregation gains and that our approach is scalable. Handling the demands within an ISPs aggregation network of multiple 100,000 of customers should be possible

5.5 Impact of misprediction

So far we presume that we have perfect knowledge about the future traffic. However, this is unrealistic and although traffic popularity is consistent with a Zipf-like distribution for any time bin, the popularity of a particular set of prefixes changes over time [34, 46, 47]. Therefore FIBIUM has to predict which prefixes are going to be popular and account for the probability it may make mistakes and miss some prefixes. To understand the impact of such misses we show how the FIB miss rate increases if one does not include the most popular prefixes in the FIB. Figure 4(b) shows the number of packets per second for the slow path if the top x prefixes in terms of traffic volume are not included in the FIB, for a FIB size of 2,000 entries. The value 0 on the x-axis corresponds to the number of slow path packets without misprediction.

When the most popular prefix is not included the penalty is on average 3,000 packets per second. When the two top prefixes are missed the penalty is 6,000. However, the increase in penalty is sub-linear. We added linear regression lines performed on the first few points of the curve to highlight this sub-linearity on the log-log scale. This implies that if one misses one of the lower ranked prefixes the penalty is not that significant. This means FIBpredict should focus on the most popular prefix. Figure 4(b) also includes the standard deviation, that reflects time of the day variations. The plot also shows the results for all four considered time bin sizes: 1, 10, 60, and 600 seconds. Note, neither the results nor their variability differ much across time bin sizes. However, the composition of the prefixes does change. This gives us the hint that any good prediction strategy should be based upon both short term popularity as well as long term popularity

However, Figure 3(b) shows that there is a difference in what time scale we consider regarding the penalty for the slow path. Indeed, this penalty increases as the FIB size increases. This is partly due to the fact that there is a strong overlap between prefixes that are among the top 100 prefixes for all considered time granularities. However, the overlap is smaller among the 100–1,000 top pre

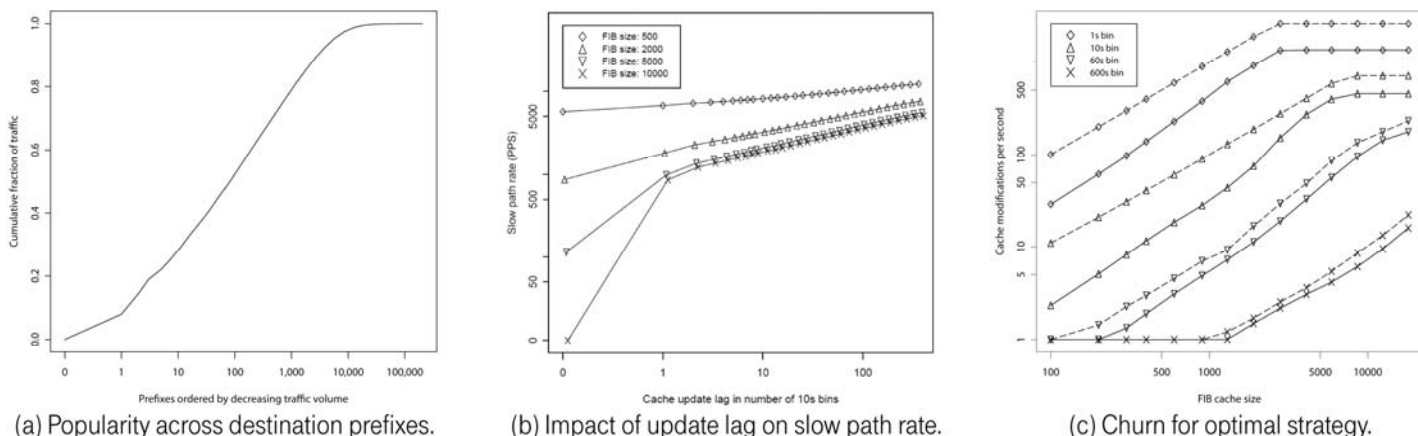


Figure 5: Traffic workload characteristics for TROUT.

fixes and even smaller among the 1,000–10,000 top prefixes. This is natural as a small local traffic spike can cause a prefix to be considered among the top 100 prefixes for 1 second bins but only among the top 1,000 prefixes for 10 second bins and not even among the top 10,000 prefixes for 600 second bins. Figure 4(c) shows how the slow path penalty is distributed across the two day period for a FIB size of 500 and 2000 entries. We note that the penalty is strongly correlated with the overall traffic volume. There are drops during the nights and a lesser drop during lunch time.

6 FIB management strategies

So far we presumed that FIB updates are instantaneous and we presumed perfect future knowledge about the traffic. However, this is not the case. Moreover, Internet traffic varies over time [6, 13], as does the set of most popular prefixes seen on a given link [34, 46, 47]. In this section, we derive a smart prefix selection strategy, called **FIBpredict**, which aims to maximize the number of packets handled by the fast path while minimizing the number of FIB updates. However, before we turn to our proposed strategy we first examine how well traditional reactive caching approaches perform.

6.1 Traditional reactive caching

Typical caching techniques are reactive: when a cache miss occurs the corresponding object is fetched and then placed into the cache. The central difference among those reactive caching techniques is which existing cache entry to remove. Accordingly, every cache miss leads to a cache modification. Our problem differs in that in general, cache misses are not resolved by a cache modification. Two common caching strategies are least recently used (LRU) and least frequently used (LFU). We implement both based on the optimistic assumption that cache replacement is instantane-

ous. Even under this very optimistic assumption, we show that traditional cache replacement algorithms have fundamental limitations.

The lines for LRU and LFU in Figure 6(a) show the FIB cache update rates vs. FIB size. We see that LRU outperforms LFU which is consistent with previous work on route caching [22]. The graph also shows the results for the optimal strategy based on 10 second bins. We see that the churn of both LRU and LFU are substantial. Indeed, based on the results from Section 4.2 they are presumably not realizable. Moreover, they are several orders of magnitude higher than the churn of optimal for reasonable sized FIBs. Only for large FIB sizes LRU becomes comparable. However, the slow path requirements of optimal cannot be improved. Figure 6(b) shows the slow path requirements for optimal, LRU, and LFU. Note, that for LRU and LFU the cache update rate is a lower bound for the slow path requirements.¹ The difference between Figure 6(a) and Figure 6(b) is that the first uses log-log scale while the second uses log-linear scale. The caching literature, e.g., [30,53], proposes to reduce churn by prioritizing both the most recent entries as well as the most stable ones over some limited time window. Our proposed strategy follows the same principle.

6.2 Traffic workload analysis for FIBpredict

We perform the selection of FIB entries not only based on FIB cache misses but on the whole set of potential destinations. Hereby, the goal is to predict the most popular prefixes while minimizing churn. To explore the trade-offs we now reexamine our workload.

¹This is because we assume FIB updates are instantaneous and therefore there are no cache misses while the FIB update is processed.

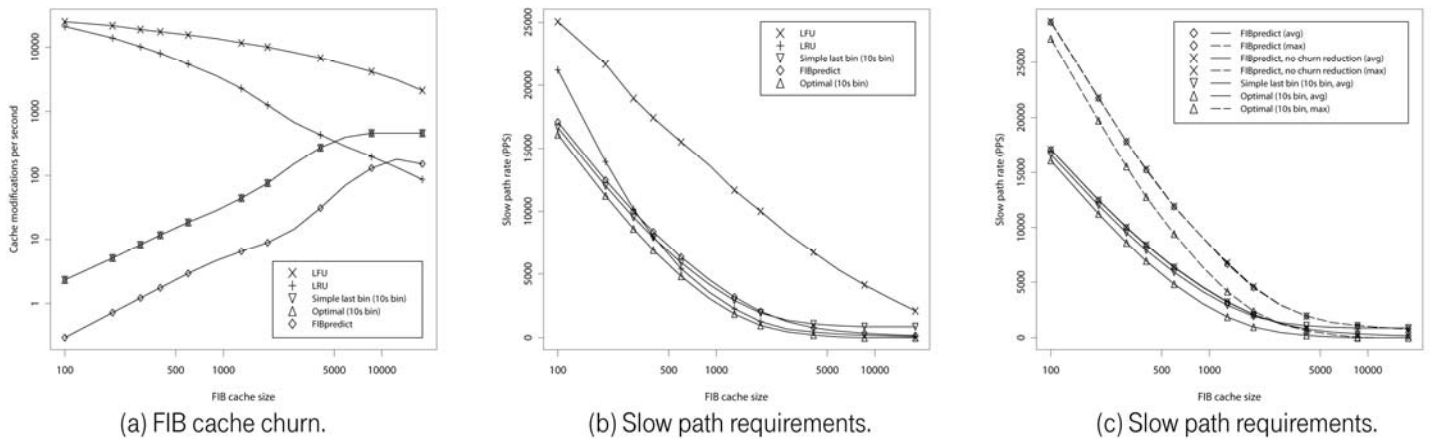


Figure 6: FIBpredict evaluation for TROUT.

The key observation has to do with the Zipf-like distribution of popular destinations in current Internet traffic, shown in a cumulative manner on Figure 5(a). However, by itself it is not sufficient since prediction becomes difficult and the churn might be prohibitive, if popular prefixes change too quickly. The simplest and most straightforward approach for selecting FIB cache entries is to concentrate on the most promising prefixes in terms of recent popularity. For example, one may choose the top N prefixes in terms of traffic volume from the last time bin. We call this selection strategy the simple one.

The traffic statistic collection and the FIB update process are not instantaneous. Therefore, we now examine the impact of this time lag on the slow path, see Figure 5(b) for 10 second time bins. First, not knowing the traffic has a penalty. For a FIB size of 500 the cache misses per second vary between 5,700 and 12,000. For a FIB size of 2,000 they vary from 870 to 2,700. However, the changes are not quite that drastic afterwards. Indeed, we observe an almost linear relationship for time lag values larger than five 10s time bins. This weak dependency between time lag and miss rate implies that the delay on the communication channel between the switch and the route controller is unlikely to cause any problems in practice.

However, FIB churn may become a burden once the FIB size grows to a few thousands of prefixes. Recall, Figure 3(c) shows the total number of FIB cache modifications required per time bin. This rate is the same independent of the lag between the traffic statistics collection and the FIB modifications. When the FIB size is small, e.g., less than 500 entries, there is little churn. However, once the FIB cache increases to include prefixes that do not correspond to significant traffic volumes, the churn increases. Figure 5(c) shows how many FIB entries have to be changed on average (solid lines) and in the worst case (dashed lines) for various FIB cache sizes and time bins. Note, the top N prefixes are very stable

for larger bin durations. Even for FIB sizes of 5,000 entries the average churn is less than 5 and the maximum less than 10 FIB entries per bin. However, for smaller bin sizes they are much more variable leading to an almost complete rewrite of the FIB.

- Recent history:** which means short duration time bins in order to reduce slow path overhead and
- Long term trends:** which means long duration time bins in order to reduce the churn rates.

The former means letting the natural variability of traffic control the content of the FIB cache. The latter means identifying the natural stability in popular destinations. Moreover, the latter is necessary to be robust against attacks (Section 6.4).

This tradeoff illustrates the difficulty of relying on simple FIB cache management techniques, e.g., LRU, LFU, or using recent summaries, and may explain why router vendors have been reluctant to rely on route caching, especially in the core of the Internet [4]. The natural variability of Internet traffic makes FIB cache management non trivial.

6.3 Multiple time scales FIBpredict

The design criteria for the multiple time scales **FIBpredict** strategies are to keep most traffic on the fast path, to minimize churn, and to make it hard for an attacker to target the slow path. To keep most of the traffic on the fast path we again rely on the Zipf property of the traffic. To keep churn low we rely on long term-trends based on traffic statistics, see Section 4.2, gathered from both the switch as well as from the slow path forwarding engine. To ensure flexibility and short reaction times, e.g., in case of traffic changes or DoS attacks, we also include recently popular prefixes in the FIB cache. This means that we are using a multiple

time scales **FIBpredict** strategy that keeps a memory of past behaviour and does not wait for FIB cache misses to occur. Indeed, cache misses in general do not trigger FIB updates.

To gather the necessary statistics we propose to keep either packet or byte counts for B different time granularities per prefix. Possible time granularities are for example the bin sizes that we considered so far: 10, 60, and 600 seconds. This implies that we gather traffic statistics every 10 seconds. Then, we simply shift the values across larger time granularities.

From each statistics vector we select a set of m top entries by number of packets or traffic volume. While one may naively expect that this results in $B * m$ entries, this is not the case as the overlap between the popular prefixes is substantial. Indeed, we typically see an overlap of more than 95% for the same time granularity and 80% for different time granularities. Therefore, if **FIBpredict** needs to select FIB entries for a FIB size of N , we propose to choose $m = N/(B-1)$, which for $N = 2000$ and $B = 3$ corresponds to $m = 1000$. The selected prefixes then are ranked by decreasing volume (traffic or packets) and we select the top m entries if there are more than m entries. If there are less than m entries we augment the FIB cache with the next most popular prefixes selected from the largest time bin, and if these cannot fill the m entries we select entries from the next most popular ones from the smaller time bins.

Next, in order to reduce churn we compare the popularity of the currently used FIB entries in reverse order with the popularity of the proposed replacements. We then only replace those FIB entries for which the popularity of the new entry is at least twice as large as the one of the old FIB entry. Figure 6(c) shows the slow path requirements vs. the FIB size for our proposed **FIBpredict** strategy. **FIBpredict** performs slightly worse than the optimal strategy which has perfect knowledge of the future. It has similar performance as the simple strategy which relies on the information of the last bin. For larger FIB cache sizes **FIBpredict** even outperforms the last bin strategy. These observations hold for both the average as well as the maximum. It is not surprising that **FIBpredict** does not do much better than the last bin strategy as they both rely on similar information.

However, in terms of FIB churn, see Figure 6(a), **FIBpredict** outperforms not only the simple strategy but also the optimal one. The reason is that neither the simple strategy nor the optimal takes long-term trends into account. As such they are not geared towards optimizing the churn. Given the churn per time bin for **FIBpredict** and our results from Section 4.2, we conclude that the communication channel will not be a bottleneck for **FIBIUM**.

Comparing **FIBpredict** with the results for LRU and LFU, see Figure 6(a) and 6(b), shows that **FIBpredict** is significantly better than LFU on both metrics. Compared to LRU, **FIBpredict** wins in terms of churn and is only slightly worse than LRU regarding slow path requirements. However, as already pointed out, we underestimate the slow path requirements for LRU.

6.4 Attacks against FIBpredict

The parts of our proposal that are most vulnerable to attacks are the slow path and the communication channel. Let us start by considering possible attacks on the slow path.

If an attacker wants to increase the load on the slow path he can do so by increasing the traffic to unpopular prefixes. His only constrain is that he has to ensure that the amount of traffic to any of the prefixes is below the amount of traffic to the least popular prefix in the FIB, i.e., that is handled by the fast path. Let us presume that the FIB contains N prefixes and handles $T * x/100$ Gbps which corresponds to $x\%$ of overall traffic T . Typically, we will choose N such that x is well larger than 90%. Moreover, the popularity of the least popular entry in the FIB is ρN which is much smaller than $T/(N * \alpha N)$ where $\alpha > 0$ is a parameter of the Zipf distribution. If the attacker wants to double the traffic on the slow path he has to add 10% extra traffic. For this he has two choices: either he distributes the traffic evenly among at least $T * 0.1/\rho N$ prefixes of volume less than ρN or he distributes traffic consistent with Zipf's law to an even larger number of prefixes but in such a way that the largest contribution is less than ρN . Fortunately, the latter is very difficult as the traffic quantities become negligible very fast. The former is also not very practical, as it disturbs the standard Zipf like distribution and therefore this attack can be detected by **RouteVisor** that has a global view of the traffic statistics.

If the attacker wants to target the communication channel he has to add additional churn to it. One way to do this is to temporally increase the popularity of prefixes for the smallest considered time bins. While this is certainly possible for some period of time, **RouteVisor** can monitor the update rate and if it notices that the effectiveness of popularity prediction of any of the frequency values decreases significantly, it can eliminate the corresponding entries temporarily from the **FIBpredict** strategy. This brings us back to being robust to attacks. Moreover, if **RouteVisor** notices certain abnormal popularity shifts, it may decide to redirect such traffic to a Network Intrusion Detection System for further analysis. It is certainly possible to install protective FIB entries for this purpose.

7 Related work

Issues in the scalability of the existing routing architecture have led to several recent proposals: NIRA [49] suggests to change the interdomain routing architecture to make it more scalable. Another way to scale routing is to increase the lifetime of existing routers, as proposed by ViAggre [2]. Finally, scaling enterprise networks is also critical as large layer-2 networks suffer from significant problems [23]. **FIBIUM** differs from those architectural approaches in that we do not require any modification to the routing architecture. Similarly to ViAggre, we try to make better

FIBIUM:

Towards Hardware Accelerated Software Routers

use of the capabilities of existing network equipment. However, we do not target the lifetime of existing routers, but propose a new way to leverage the capabilities of less expensive commodity switches to transform them into routers.

Making IP forwarding fast and scalable has been and still is one of the major problems of network equipment vendors. Early IP routers were performing forwarding in software. As the Internet grew, routing tables were becoming large enough to require specialized data structures to limit the size of forwarding tables and allow efficient longest-prefix matching at the same time [8, 45].

Recent work has argued about today's relevance of route caching [22,50]. In [22], the authors propose to keep a cache for the most popular prefixes. Based on traffic traces from an ISP network, the authors of [22] evaluate the performance of route caching by measuring the miss rate in percentage of the total traffic. They show that this miss rate is reasonably low. In [50], the authors propose to rely on Bloom filters in fast memory to scale the forwarding in enterprise networks. They find that such an approach has serious limitations under worst-case workloads.

Leveraging fast memory for forwarding is one of the central issues of scalable forwarding [48]. Numerous papers have proposed the use of TCAM for forwarding purposes [24, 41, 51]. Current TCAMs allow very efficient longest prefix matching but have limitations, e.g., power consumption and price. Router vendors nowadays rely on different types of memories on their line cards to ensure a high performance-cost ratio [48].

8 Summary and future work

In this work we explore how to take advantage of the significant capabilities of commodity switches, open source software routing, and PC based routers to build a hardware accelerated software router alternative, called **FIBIUM**. Our prototype, which is based on Quagga, OpenFlow, and our controller **RouteVisor**, is capable of inter-operating with routers from multiple vendors and its performance is sufficient to handle full IP routing tables and the traffic requirements of a carrier aggregation network. In addition, we propose a prefix selection strategy, called **FIBpredict**, that takes advantage of the capabilities of the TCAM-based FIB of current switches for fast path forwarding. The strategy we propose outperforms traditional caching strategies and maximizes fast path forwarding while keeping the communication overhead low.

In future work we are planning to further improve the software base by extending the supported features, e.g. IP multicast. We are also investigating the applicability of **FIBIUM** to different deployment scenarios, e.g. data centers.

9 Table of figures

Figure 1: Architectural overview of current IP routers and FIBIUM.	3
Figure 2: FIBIUM design with its slow path, fast path, and communication channel.....	5
Figure 3: Baseline results for TROUT.....	8
Figure 4: Baseline Scalability and misprediction.	9
Figure 5: Traffic workload characteristics for TROUT.	10
Figure 6: FIBpredict evaluation for TROUT.....	11

10 References

- [1] ARISTA NETWORKS. EOS: An extensible operating system. www.aristanetworks.com/en/EOS, 2009.
- [2] BALLANI, H., FRANCIS, P., CAO, T., AND WANG, J. Making routers last longer with viaggre. In NSDI (2009).
- [3] BEN HOUIDI, Z., MEULLE, M., AND TEIXEIRA, R. Understanding slow bgp routing table transfers. In ACM IMC (2009).
- [4] BOLLAPRAGADA, V., MURPHY, C., AND WHITE, R. Inside Cisco IOS Software Architecture. Cisco Press, 2000.
- [5] CISCO. NetFlow services and applications. <http://www.cisco.com/warp/public/732/netflow>, 1999.
- [6] CLAFFY, K. C., AND BROWNLEE, N. Understanding Internet traffic streams: Dragonflies and Tortoises. IEEE Communications Magazine (2002).
- [7] CLAISE, B. Specification of the IP flow information export (IPFIX) protocol for the exchange of IP traffic flow information. RFC5101, 2008.
- [8] DEGERMARK, M., BRODNIK, A., CARLSSON, S., AND PINK, S. Small forwarding tables for fast routing lookups. In ACM SIGCOMM (1997).
- [9] DOBRESCU, M., EGI, N., ARGYRAKI, K., CHUN, B., FALL, K., IANNACCONI, G., KNIES, A., MANESH, M., AND RATNASAMY, S. RouteBricks: exploiting parallelism to scale software routers. In ACM SOSP (2009).
- [10] EDWARDS, J. Enterprises cut costs with open-source routers. <http://www.computerworld.com/s/article/9133851>, 2009.
- [11] EGI, N., GREENHALGH, A., HANDLEY, M., HOERDT, M., HUICI, F., AND MATHY, L. Towards high performance virtual routers on commodity hardware. In ACM CoNEXT (2008).
- [12] FANG, W., AND PETERSON, L. Inter-as traffic patterns and their implications. In IEEE Global Internet (1999).
- [13] FELDMANN, A., GILBERT, A., HUANG, P., AND WILLINGER, W. Dynamics of IP traffic: a study of the role of variability and the impact of control. In ACM SIGCOMM (1999).
- [14] FELDMANN, A., KONG, H., MAENNEL, O., AND TUDOR, A. Measuring BGP pass-through times. In PAM (2004).
- [15] FOULI, K., AND MAIER, M. The road to Carrier-Grade Ethernet. IEEE Communications Magazine (2009).
- [16] HALABI, B., AND PHERSON, D. M. Internet Routing Architectures (2nd Edition). Cisco Press, 2000.
- [17] HANDLEY, M., HODSON, O., AND KOHLER, E. Xorp: an open platform for network research. ACM CCR 33, 1 (2003).
- [18] HUSTON, G. BGP Routing Table Analysis Reports. <http://bgp.potaroo.net/>.
- [19] Ip Infusion ZebOS. <http://www.ipinfusion.com/>.
- [20] J. REXFORD, J. WANG, Z. X., AND ZHANG, Y. Bgp routing stability of popular destinations. In ACM IMC (2002).
- [21] JAIN, R. Characteristics of destination address locality in computer networks: A comparison of caching schemes. Computer Networks and ISDN (1989/90).
- [22] KIM, C., CAESAR, M., GERBER, A., AND REXFORD, J. Revisiting route caching: The world should be flat. In PAM (2009).
- [23] KIM, C., CAESAR, M., AND REXFORD, J. Floodless in seattle: a scalable ethernet architecture for large enterprises. In ACM SIGCOMM (2008).
- [24] KOCAK, T., AND BASCI, F. A power-efficient tcam architecture for network forwarding tables. J. Syst. Archit. (2006).
- [25] KOHLER, E., MORRIS, R., CHEN, B., JANNOTTI, J., AND KAASHOEK, F. The Click modular router. ACM Trans. Comput. Syst. 18, 3 (August 2000), 263–297.
- [26] LABOVITZ, C., LEKEL-JOHNSON, S., MCPHERSON, D., OBERHEIDE, J., AND JAHANIAN, F. Internet Inter-Domain Traffic. In ACM SIGCOMM (2010).
- [27] MAENNEL, O., AND FELDMANN, A. Realistic BGP traffic for test labs. In ACM SIGCOMM (2002).
- [28] MAIER, G., FELDMANN, A., PAXSON, V., AND ALLMAN, M. On dominant characteristics of residential broadband internet traffic. In ACM IMC (2009).
- [29] 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).
- [30] MEGIDDO, N., AND MODHA, D. Arc: A self-tuning, low overhead replacement cache. In USENIX Conf. on File and Storage Technologies (2003).

FIBIUM:

Towards Hardware Accelerated Software Routers

- [31] MOGUL, J., YALAGANDULA, P., TOURRILHES, J., MCGEER, R., BANERJEE, S., CONNORS, T., AND SHARMA, P. API design challenges for open router platforms on proprietary hardware. In ACM HotNets (2008).
- [32] MOY, J. OSPF: Anatomy of an Internet Routing Protocol. Addison-Wesley, 1998.
- [33] NAOUS, J., ERICKSON, D., COVINGTON, G., APPENZELLER, G., AND MCKEOWN, N. Implementing an OpenFlow switch on the NetFPGA platform. In ACM ANCS (2008).
- [34] PAPAGIANNAKIT, K., TAFT, N., AND DIOT, C. Impact of flow dynamics on traffic engineering design principles. In IEEE INFOCOM (2004).
- [35] PAXSON, V. Bro: a system for detecting network intruders in real-time. Computer Networks (1999).
- [36] PHAAL, P., PANCHEN, S., AND MCKEE, N. InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks. RFC3176, 2001.
- [37] Quagga Routing Suite. <http://www.quagga.net>.
- [38] REXFORD, J., WANG, J., XIAO, Z., AND ZHANG, Y. BGP Routing Stability of Popular Destinations. In ACM IMW (2002).
- [39] RIPE routing information service. <http://www.ripe.net/ris/>.
- [40] University of Oregon Route Views Project. <http://www.routeviews.org/>.
- [41] SHAH, D., AND GUPTA, P. Fast incremental updates on Ternary-CAMs for routing lookups and packet classification. IEEE Micro (2001).
- [42] SHYU, W., WU, C., AND HOU, T. Efficiency analyses on routing cache replacement algorithms. In IEEE ICC (2002).
- [43] SIMCOE, R. J., AND PEI, T.-B. Perspectives on ATM switch architecture and the influence of traffic pattern assumptions on switch design. ACM CCR (1995).
- [44] STRINGFIELD, N., WHITE, R., AND MCKEE, S. Cisco Express Forwarding. Cisco Press, 2007.
- [45] WALDVOGEL, M., VARGHESE, G., TURNER, J., AND PLATTNER, B. Scalable high speed IP routing lookups. In ACM SIGCOMM (1997).
- [46] WALLERICH, J., DREGER, H., FELDMANN, A., KRISHNAMURTHY, B., AND WILLINGER, W. A methodology for studying persistency aspects of internet flows. ACM CCR (2005).
- [47] WALLERICH, J., AND FELDMANN, A. Capturing the variability of internet flows across time. In IEEE Global Internet (2006).
- [48] WHITTLE, R. SRAM-based IP forwarding eliminates the need for route aggregation. Internet draft, draft-whittle-sram-ip-forwarding-01.txt, work in progress, October 2007.
- [49] YANG, X., CLARK, D., AND BERGER, A. NIRA: a new inter-domain routing architecture. IEEE/ACM Trans. Netw. (2007).
- [50] YU, M., FABRIKANT, A., AND REXFORD, J. BUFFALO: bloom filter forwarding architecture for large organizations. In ACM CoNEXT (2009).
- [51] ZANE, F., NARLIKAR, G., AND BASU, A. CoolCAMs: power-efficient TCAMs for forwarding engines. In IEEE INFOCOM (2003).
- [52] ZHANG, Y., BRESLAU, L., PAXSON, V., AND SHENKER, S. On the characteristics and origins of internet flow rates. In ACM SIGCOMM (2002).
- [53] ZHOU, Y., PHILBIN, J., AND LI, K. The multi-queue replacement algorithm for second level buffer caches. In Usenix (2001).

FIBIUM:
Towards Hardware Accelerated Software Routers

Publisher:

Deutsche Telekom AG
Laboratories
Ernst-Reuter -Platz 7
D-10587 Berlin
Telefon: +49 30 8353-58555
www.laboratories.telekom.com

Authors: Nadi.Sarrar@telekom.de
Anja.Feldmann@telekom.de
Steve.Uhlig@telekom.de
R.Sherwood@telekom.de
Xin.Huang@telekom.de

© 2010 Deutsche Telekom Laboratories

The information contained in this document represents the current view of the authors on the issues discussed as of the date of publication. This document should not be interpreted to be a commitment on the part of Deutsche Telekom Laboratories, and Deutsche Telekom Laboratories cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. Deutsche Telekom Laboratories makes no warranties - express, implied, or statutory - as to the information in this document.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording or otherwise), or for any purpose, without the express written permission of Deutsche Telekom Laboratories.

Deutsche Telekom Laboratories may have patents, patent applications, trademarks, copyrights or other intellectual property rights covering the subject matter in this document. Except as expressly provided in any written license agreement from Deutsche Telekom Laboratories, the furnishing of this document does not give you any license to these patents, trademarks, copyrights or other intellectual property.