A Flexible Quagga-Based Virtual Network with FIB Aggregation

Jerald Paul Abraham, Yaoqing Liu, Lan Wang, and Beichuan Zhang

Abstract

The rapid increase in routers' forwarding table size is raising serious concerns for ISPs. In particular, it exhausts the routers' forwarding hardware capacity, leading to more frequent upgrades and higher cost. In this article, we present the development of a virtual network framework based on open source software that demonstrates how solutions to this impending problem can be implemented and studied. The system is capable of emulating an operational network environment with intra-domain and inter-domain routing protocols as well as real-world Internet routing traffic. By adding performance monitoring and FIB aggregation capabilities to this system, we are able to evaluate the performance of FIB aggregation algorithms in a realistic network environment.

With the Internet's explosive growth, the global routing table size has been increasing very quickly. This rapid growth causes serious concerns: Internet service providers (ISPs) have to upgrade routers and switches more frequently in order to keep up with the increasing routing table size, leading to higher cost in running ISP networks. We have developed forwarding information base (FIB) aggregation algorithms that can compress routers' FIB considerably [1–3] and achieve the same forwarding behavior as the uncompressed FIB. Our implementation of the algorithms can reduce the FIB size by 40–70 percent with minimal processing overhead. Moreover, FIB aggregation can be deployed by a software update on routers and switches without any hardware change. Discussions with many network operators revealed the present state of this problem, and we obtained a clear message that FIB aggregation has great commercial utilization potential. However, our standalone implementation is not convincing enough — we need to integrate our implementation with a real software or hardware router, and demonstrate its effectiveness quantitatively under a realistic network environment.

To convince network operators and router vendors, we decided to build a real-time demonstration system of FIB aggregation. However, we face several challenges. First, routers from commercial companies have closed source software that we cannot modify. Second, we do not have access to operational networks to deploy our implementation. Considering all these factors, we built a virtual network system using solely open source software including Quagga [4], VirtualBox [5], and BGPSimple [6]. More specifically, we implemented one of our FIB aggregation algorithms [3] in Quagga, a widely used open source routing software suite. We then used VirtualBox to build a virtual network consisting of many virtual machines, each running an instance of Quagga. The virtual network can be configured to emulate any network topology and can be reconfigured easily. Finally, we fed real-time routing data from the BGPMon project [7] and the RouteViews project [8] into the virtual routers. Through this system we are able to monitor the virtual routers' FIB size and CPU usage, which demonstrate the performance of our FIB aggregation algorithms under a realistic network environment.

In addition to using the system within our research team, we can give access to interested network operators so that they can log into the virtual routers and check the performance in real time by themselves. We can also peer with interested ISPs to receive and process their routing updates so that the virtual routers will reflect what the ISP routers would have if they run the FIB aggregation algorithm. Furthermore, the system can be used to test other improvements to the routing system. We plan to publish our code and scripts so that others can use this platform to evaluate their design and implementation.

The remainder of the article is organized as follows. First, we give relevant background information and present a system overview. We then describe the implementation of FIB aggregation in Quagga. Next, we present our methodology for emulating an ISP network and our experimental results. Then we discuss related work and conclude our work.

Background

Routers are critical network equipment in the Internet infrastructure. They forward data traffic in a hop-by-hop fashion. At each hop, the router uses a FIB, also called a forwarding table, to store the next hop of each address prefix. A router typically have multiple copies of its FIB, each residing on a line card connecting the router to one of its neighbors. This way, when a line card receives a packet, it can consult its local FIB in order to forward the packet, instead of having to look up a central FIB, which can easily become a performance bottleneck.

Jerald Paul Abraham and Beichuan Zhang are with the University of Arizona.

Yaoqing Liu is with Clarkson University.

Lan Wang is with the University of Memphis.

 0890-8044/14/$25.00

(a) Original FIB entries		
Label	**Prefix**	**Next hop**
A	141.225.0.0/16	1
B	141.225.64.0/18	1
C	141.225.32.0/19	1
D	141.225.96.0/19	2
E	141.225.48.0/20	2
(b) Aggregated FIB entries		
Label	**Prefix**	**Next hop**
A	141.225.0.0/16	1
D	141.225.96.0/19	2
E	141.225.48.0/20	2

Table 1. FIB entries before and after aggregation.

Since the FIB contains information about how to reach all the networks on the Internet, its size increases as the Internet grows. As an example, Fig. 1 shows the super-linear growth of the routing table size from 1989 to 2013 observed in Telstra, Australia's largest ISP [9]. The same trend is happening in all ISP networks. This problem is widely recognized by the network operation and standardization community (RFC 4984). They are mainly concerned about the cost of upgrading line cards to accommodate larger FIB tables. For high forwarding performance, today's routers store FIB in fast memory such as TCAM or SRAM, which is much more expensive than regular memory such as DRAM. According to Fall *et al.* [10], "SRAM density is at least one order of magnitude less than DRAMs while cost can be two orders of magnitude greater."

While scaling the FIB is important to sustaining the growth of the Internet, a major concern of the ISPs is the deployment cost of the potential solution since it may require changes in routing protocols, coordination among routers or ISPs, or even changes in end-user computers.

FIB aggregation combines multiple FIB entries into one without losing any useful information. This means that the new table should not change the next hops that the packets take to reach their destination. This approach leverages the hierarchical organization of prefixes and exploits opportunities when multiple forwarding table entries can be represented as one. For instance, when several consecutive prefixes share a common next hop, they can be combined into a shorter prefix with the same next hop. Table 1 illustrates this case. Another case is when a prefix and its nearest ancestor prefix share the same next hop; this prefix can be removed from the FIB. In both cases, the longest prefix match will return the shorter prefix, but the returned next hop will still be correct. There are more complex cases where aggregation can be applied. Our FIFA aggregation algorithm is an improvement over Optimal Routing Table Constructor (ORTC), and the specifics of our approach are discussed in great detail in [2, 3].

FIB aggregation is a promising approach as it not only reduces the FIB size significantly but also has low deployment cost. The deployment can be done by a software update on routers and switches, and ISPs can deploy it incrementally in their networks, one router/switch at a time. Moreover, it does not affect any network communication protocols. There is no need to change anything in network operations such as monitoring and traffic engineering.

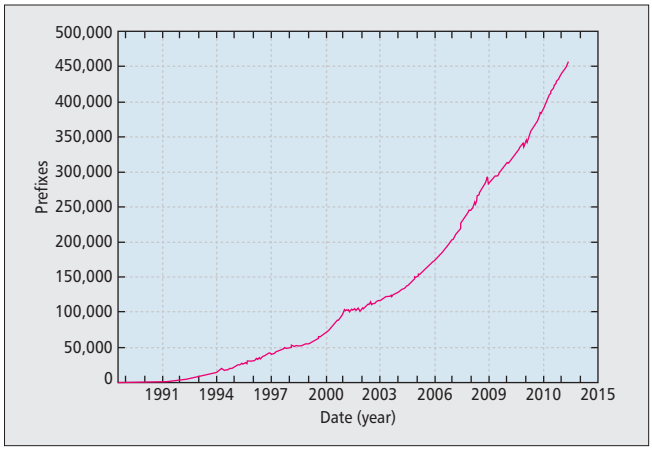

Figure 1. Global BGP routing table size observed at Telstra.

Design

Our overall goal is to build a prototype system to demonstrate the performance of FIB aggregation and serve as a reusable framework for the analysis of improvements to the Internet routing system. Figure 2 gives an overview of our system. The BGPMon [7] server at Colorado State University provides real-time Border Gateway Protocol (BGP) updates to the system. The stream converters establish telnet connections to BGPMon, receive BGP XML streams, and convert the streams into BGP tabledump format to further feed the BGP speakers. The BGP speakers establish eBGP sessions with Quagga's BGP daemons and send standard BGP messages to the virtual ISP network. Each virtual machine (VM) within the virtual network runs Quagga daemons (Open Shortest Path First, OSPF, and BGP in particular) with or without FIB aggregation. In addition, each VM runs a live performance monitoring tool to collect performance metrics and a python Log Parser to parse logs. A web server collects all of the parsed logs and utilizes the Asynchronous JavaScript and XML (AJAX) technique to update a client web page with new content without refreshing the whole page.

Network Virtualization

As mentioned previously, we do not have direct access to an operational network, so we resort to emulating such a network through virtualization. Ideally, the virtual network can easily emulate any desired network topology. One option is to use an existing shared virtualization resource such as Emulab. The other is to use open source virtualization software to create this network ourselves. We need a continuously running system that can emulate a large number of routers, but a shared virtualization environment may not have enough resources and may not allow the experiment to run for a long period of time. Therefore, we decided to adopt the second option and use VirtualBox for our system. VirtualBox is a GNU Licensed open-source software that enables creation of high performance virtualization environments using powerful servers. Through VirtualBox commands, we create VMs that emulate routers as well as the communication links between the routers based on a desired network topology.

Routing Software

We need routing software to transform each VM into a full-fledged router. This software should support major routing protocols and allow us to configure the routers easily. We chose Quagga based on these considerations.

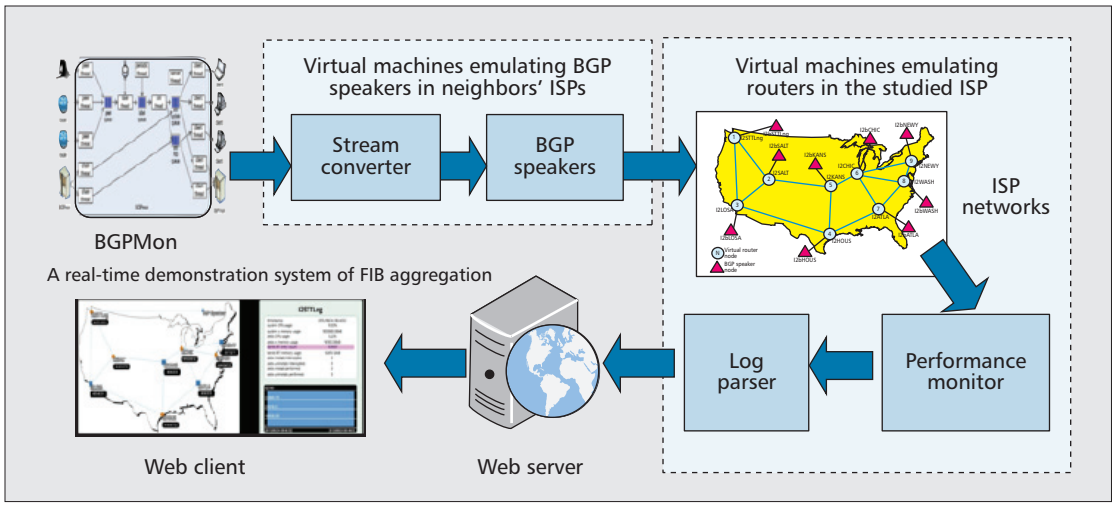

Figure 2. System overview.

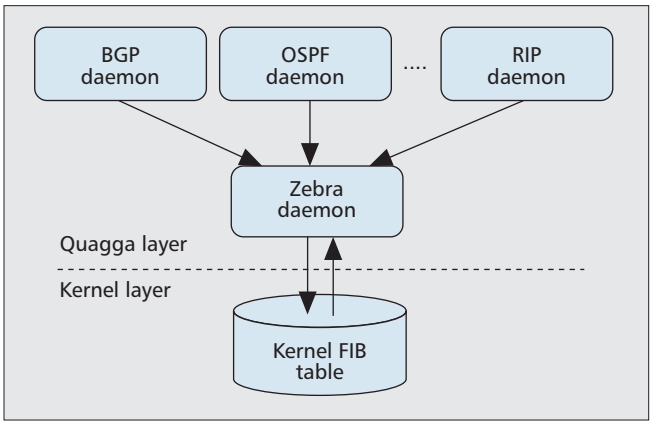

Figure 3. Quagga modules without FIB aggregation.

Quagga [4] is a widely used open source routing software. It supports a variety of network protocols including OSPFv2, OSPFv3, RIPv1, RIPv2, RIPng, and BGP-4; the software is available for UNIX platforms, specifically FreeBSD, Linux, Solaris, and NetBSD. Originally forked out of GNU Zebra, Quagga's design abstracts the various protocol implementations as separate daemons that could be run individually on top of the core daemon, Zebra. These daemons process the protocol-specific routing updates and direct the forwarding table modifications to the Zebra daemon. The Zebra daemon then processes the received FIB changes and applies them onto the kernel FIB. Figure 3 illustrates a high-level layout of this workflow in Quagga. It supports a variety of routing protocols and is actively used by small networks with a configuration interface similar to commercial routers. In Quagga, we added two modules, a FIB aggregation module and an interception module, to convert an uncompressed FIB to a compressed FIB.

External Speaker

Quagga does not accept live BGP update streams or archived BGP data as its input, so we need to find a lightweight BGP speaker that can accept such input and announce BGP routes to the virtual network. We compared several open source BGP speakers that can work with Quagga, including ExaBGP, BRID, BGPfeeder, BGPSimple, and pyBGP. These applications differ in execution performance, implementation, and input feed format. We selected BGPSimple [6] as it accepts input update feeds in the most commonly available formats. It offers fast and efficient execution performance and also comes with several filtering options that enable customizable BGP update generation. We modified its code to slow down the rate of BGP update generation when reading an initial table dump file so as to enable lossless reception by the BGP peer.

Performance Monitoring

Quagga offers a wide variety of configuration options for each protocol daemon and has a good command line interface for each of them. However, we cannot acquire performance metrics such as CPU and memory usage through these features. Additionally, we need to monitor the performance metrics continuously over a period of time, so they must be logged at periodic intervals. Hence, we implemented our own performance monitoring module, which captures and logs performance metrics periodically. This module is designed to be installed as a command providing easy access for end users working with Quagga. The monitoring process can be configured for a variety of performance metrics and for their logging at desired acquisition intervals.

Adding Route Aggregation to Quagga

The first objective of our work is to add FIB aggregation and performance monitoring to Quagga without major code changes and negative performance impact. This ensures that we do not inadvertently change Quagga's implementation logic, and that our new features do not slow down the core routing and forwarding functionality. These considerations led to the updated Quagga system depicted in Fig. 4.

FIB Aggregation

In order to aggregate the FIB without changing Quagga significantly, we keep our FIB aggregation code out of the existing Quagga code. We add an interception module to intercept every FIB update after it is received by the Zebra daemon and pass all of them to the FIB aggregation module (Fig. 4). Once the interception module receives the aggregated FIB changes back from the FIB aggregation module, it applies them to the kernel FIB. Below we describe these two modules in detail.

The interception module's major functionality includes fetching static routes, interception of inbound FIB updates, application of aggregated FIB updates to the kernel FIB, and logging. The FIB updates fall into two categories:

- *Route installs*: adding one or more entries to the FIB for a new address prefix

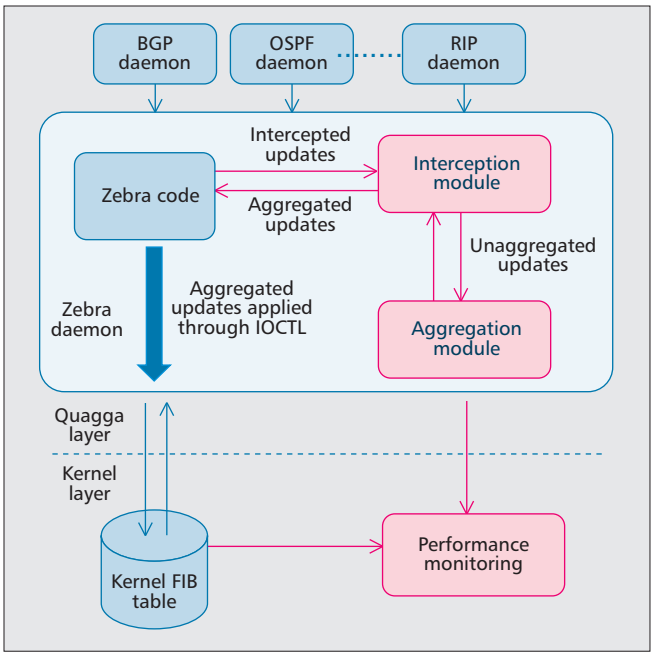

Figure 4. Quagga modules with FIB aggregation.

• *Route uninstalls*: deleting one or more entries from the FIB for an existing address prefix

The aggregation module interacts directly with the interception module but not with other parts of Quagga. It houses some temporary data structures that are needed to perform effective and incremental table aggregation. It currently encapsulates the logic of our FIFA-S algorithm [3], which can easily be replaced with any other FIB aggregation algorithm. Below we highlight several implementation issues that we did not encounter in our previous standalone implementation.

Processing static routes: When we plugged in the aggregation module to Quagga, we found a few duplicate route prefixes with different next hop values. It turned out that static routes were preconfigured during Quagga's startup. Our solution is to introduce these static routes into the aggregation module but not to aggregate them. More specifically, we set particular flags for these routes so that they are not aggregated with other routes.

Updating kernel FIB: Without aggregation, one route change will generate one kernel FIB change. With aggregation, however, one route change may lead to zero or more FIB changes because we have to remove redundant routes from FIB to keep it compressed all the time. We also learned an important fact: an update with a next hop value change actually consists of two FIB operations: Quagga first withdraws the prefix with the old next hop value and then injects the same prefix with a new next hop value.

Handling recursive next hop: The FIB entries propagated inside Quagga sometimes have next hops that are "recursive" in nature. This means the next hop to reach a destination does not refer to a direct neighbor but to another intermediate node that lies on the path to the destination. For example, the recursive next hop could be the iBGP neighbor that is closest to the destination (among all the iBGP neighbors of the node), but may be several IP hops away from this router. It is therefore necessary to look up the address of the recursive next hop in the FIB (one or more times) until we resolve this recursive next hop to a directly connected next hop neighbor. We did not realize the usage of recursive next hop in Quagga initially and had difficulty understanding why the aggregated FIB was incorrect in some cases. In fact, the Quagga route data structure includes a recursive next hop flag to determine which data field (regular next hop or recursive next hop) to use for the next hop value. Since our aggregation module originally accepts and returns only non-recursive next hop values, we had to modify our process to handle such recursive updates.

Performance Monitoring

When adding FIB aggregation to routing software, we expect the changes to result in as little CPU overhead as possible and at the same time improve other performance metrics such as memory usage. The impact of our changes can only be quantified through low-level performance metrics captured at runtime. Some metrics pertain to the overall Quagga system performance, such as CPU usage and routing process' virtual memory usage. Other metrics are specific to FIB aggregation, such as FIB entry count, FIB memory usage, and number of updates made to the FIB.

The primary statistics captured by the performance monitoring module include:

- CPU and virtual memory usage of the OS, BGP daemon, and Zebra daemon
- Kernel FIB's memory usage
- Kernel FIB entry count
- Number of route installs and route uninstalls intercepted and actually performed

Part of the module is implemented within the Zebra daemon to support capturing of statistics on Zebra's updates to the Kernel FIB.

We capture the memory and CPU utilization metrics through periodic operations on the files that the Unix kernel maintains in its proc directory. Similarly, some metrics related to the kernel's FIB are computed from the internal kernel file /proc/net/route, which holds the kernel's FIB. In addition, we capture routing update statistics in terms of route installs and route uninstalls by the addition of some program variables within the Zebra daemon. All the information gathered by the monitoring application is made available through command line options and log files.

Emulating an ISP Network

The second objective of our work is to set up a realistic *network environment* to demonstrate the performance of our FIB aggregation algorithm. To this end, we use VirtualBox, BGPSimple, and real ISPs' routing data to emulate a real operational network. Note that our methodology can be used to emulate any network, but in this section, we use the Internet2 Abilene topology as an example to illustrate our methodology. Figure 5 shows our virtual network, which includes a total of nine routers in the Internet2 topology and nine external BGP speakers belonging to other ISPs that exchange routing information with Internet2. Each node is on a separate VM. Each Internet2 router runs either the original Quagga or our updated version of Quagga (specifically the Zebra, OSPF, and BGP daemons). Each BGP speaker runs the BGPSimple program that sends real routing updates to the BGP peer.

We use OSPF to establish intra-domain connectivity among all of the Internet2 nodes. The BGP daemons on each Internet2 node then use this base connectivity to establish BGP peering sessions with their iBGP peers. For simplicity, we use full-mesh iBGP peering; that is, every router has an iBGP session with every other router in the same network. For a larger topology, we can use route reflectors to reduce the number of iBGP sessions.

In order to attain equivalence between the virtual net-

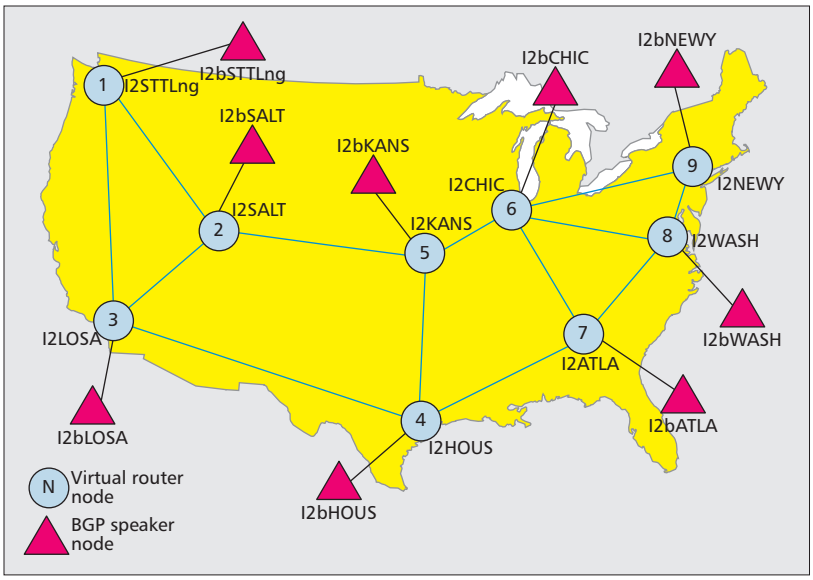

Figure 5. Virtual network setup.

work and a real-world ISP, we need the presence of a huge volume of BGP routing updates in the Internet. Fortunately, BGP routing tables and updates from many ISPs are being collected by a number of projects. We use BGP data from the RouterViews project of the University of Oregon [8] and the BGPMon project of Colorado State University [7]. The RouterViews project provides BGP table dumps and update files archived at regular intervals while the BGPMon project provides live BGP updates in addition to archived BGP data.

We modified the BGPSimple program to pump the BGP data into the network. The program first reads a routing table from a BGP table dump file (from RouteViews or BGPMon) and then sends the resulting BGP updates over its BGP session so that the peer can establish an initial BGP table. Since there are hundreds of thousands of entries in a full routing table, we added pacing to the BGPSimple program so that the peer does not become overwhelmed by the initial influx of updates. The program then sends subsequent BGP updates as they become available. These updates are obtained from either the BGP update files downloaded from RouteViews or a live stream of BGP updates from BGPMon. In order to evaluate the performance of our system under high routing load, we used large routing tables with 450,000 prefixes on average.

Evaluation

To estimate the performance impact of FIB aggregation, we ran Quagga with and without aggregation at each virtual router in two separate trials. Both trials used the same configuration and BGP feed to maintain consistency in network condition. The performance metrics were captured by our monitoring module on a node-by-node basis every 3 s, and the data was saved to logs for post analysis. It is also possible through command line options for an end user to get live performance stats during system runtime. All the experiments are performed on a single server with Dual Intel Xeon E5-2680 CPU (2.70 GHz), 128 Gbytes memory, and a 4 Tbyte disk. The server runs the 64-bit Ubuntu Server 12.04 operating system. The results are similar for all the routers, so we show the results for one of them for brevity.

Figure 6 shows the FIB size as a router processes routing updates (the X-axis is the sequence number of the BGP update). The trial without aggregation caused the kernel FIB to rise to a total of 480,000 entries and 60 Mbytes in size. With aggregation, however, the FIB size rose to around 140,000 entries and 18.5 Mbytes. This means FIB aggregation led to about 70 percent reduction in FIB size (both entry count and memory usage).

Figure 7 shows how many route installs and route uninstalls were performed on the kernel FIB. The system with aggregation performed slightly *fewer* route installs than the system without aggregation. Meanwhile, the former had slightly more route uninstalls than the latter. Consequently, the total number of FIB changes with aggregation (not shown in the figure) is only 7.1 percent higher than that without aggregation. The increased number of route uninstalls is necessary to remove certain redundant FIB entries for keeping the FIB table compact and correct. One concern regarding the increased number of FIB changes is that the overhead may lead to delays in applying routing updates to FIB on time. However, we note that the increase in FIB changes is small (7.1 percent). Furthermore, we confirmed from router vendors that FIB updates are handled in bulk mode; that is, FIB updates are not handled one by one. Instead, they are processed and applied to FIB in bulk after a certain time or a certain size of accumulated updates. This would further reduce the latency impact caused by FIB aggregation.

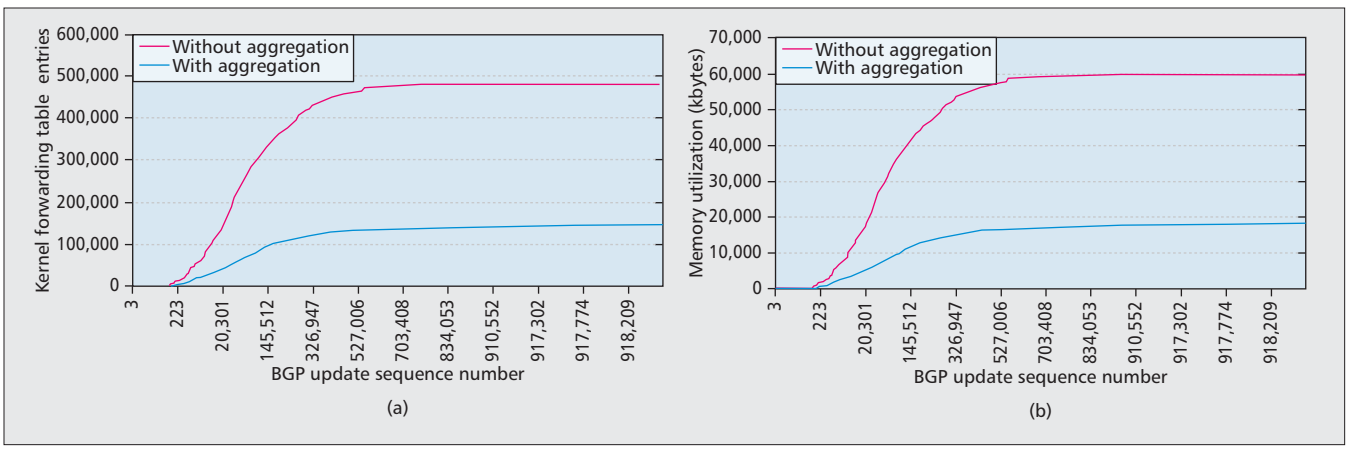

Figure 6. FIB size statistics: a) FIB entry count; b) FIB memory usage.

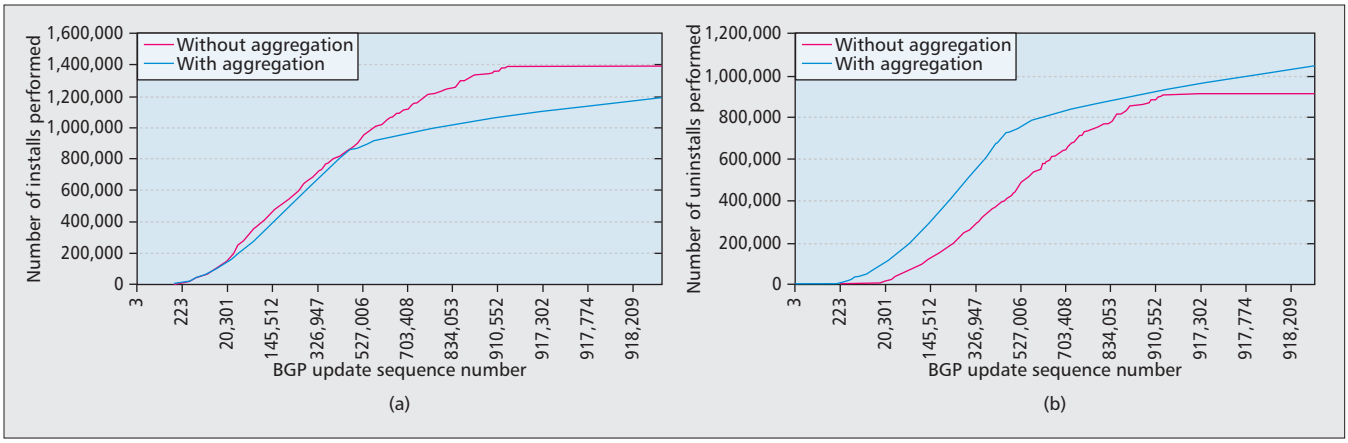

Figure 7. FIB change statistics: a) route installs on FIB; b) route uninstalls on FIB.

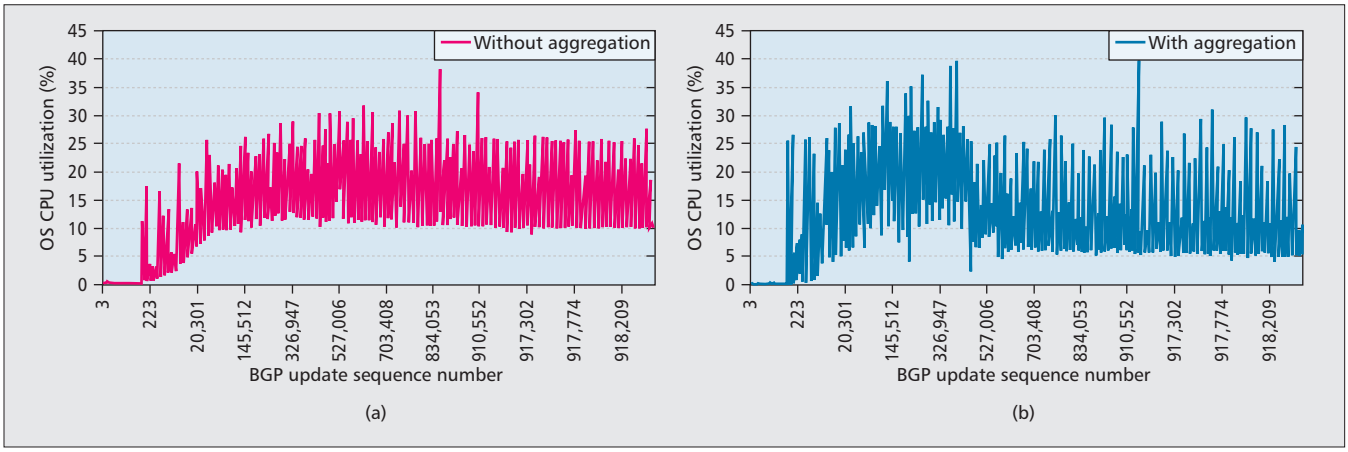

Figure 8. Operating system CPU utilization.

Figure 8 shows the CPU utilization of the overall operating system over time. It is surprising that the average OS CPU utilization is 52.7 percent lower with aggregation (5.29 percent with aggregation and 11.18 percent without aggregation). However, the average Zebra CPU utilization (not shown in the figure) is 59.4 percent higher with aggregation (1.22 percent with aggregation and 0.76 percent with aggregation). This is expected as Zebra runs the aggregation algorithm. The exact cause of the lower operating system (OS) CPU utilization with aggregation is still under investigation, but we suspect that it is due to the lower cost for the kernel to maintain a much smaller FIB table (e.g., the CPU time required to insert/delete an entry is much lower when the table size is small).

Figure 9 shows the virtual memory utilization at the OS level, which indicates that the aggregation-based system increases the OS memory usage by 0.3 Gbytes. Note, however, that the aggregation module in the route processor uses DRAM, while the FIB is typically stored in SRAM in line cards, which is at least two orders of magnitude more expensive than DRAM. Therefore, although we should try to reduce the memory consumed by the FIB aggregation module, the benefit offered by a 70 percent memory reduction in the line card still significantly outweighs the cost incurred by the memory increase in the route processor.

Typically, the more links a router has, the less aggregation it may get. Our previous results in [1] show that the FIB reduction ranges from 60 percent for routers with hundreds of links to 90 percent for routers with dozens of of links. Now we estimate the cost savings for routers with high and low connectivity. We assume that DRAM price is about \$1/Gb (\$8/Gbyte) [11], and SRAM price is 100 times that of DRAM (i.e., \$0.1/Mb, \$0.8/MB). For a router with 100 links and a FIB size reduction of 60 percent (36 Mbytes), the cost savings is $36 \times 0.8 \times 100 - 0.3 \times 8 = \$2877.6$. For a router with 10 links and a FIB size reduction of 90 percent (54MB), the cost savings is $54 \times 0.8 \times 10 - 0.3 \times 8 = \$429.6$. Therefore, FIB aggregation can reduce memory cost for both types of routers. Furthermore, because FIB aggregation would extend the life of a line card for a few more years, it would reduce the replacement frequency of line cards, which can cost thousands of dollars or more.

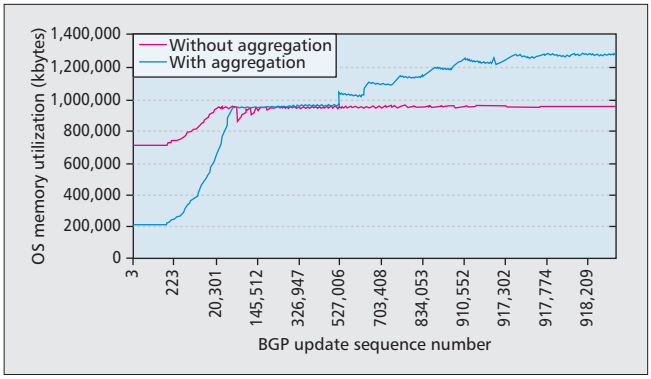

Figure 9. Operating system virtual memory utilization.

Related Work

Zhao *et al.* [12] projected that FIB caching [13] and FIB aggregation are two of the most promising solutions to reduce FIB size in the near future. The first major work on FIB aggregation is by Draves *et al.*, who proposed a one-time aggregation algorithm called ORTC [14]. More recently, Zhao *et al.* proposed an update handling algorithm for FIB aggregation [1]. Liu *et al.* proposed improved update handling algorithms based on the ORTC aggregation algorithm [2, 3]. Note that all of these FIB aggregation algorithms can be easily incorporated into Quagga according to our implementation experience.

Sarrar *et al.* ([15 16]) present traffic offloading approaches and analyze the benefit of incorporating locality in aggregation. Bienkowski *et al.* [17] present another online aggregation algorithm that aims to optimize FIB and provide asymptotic bounds on their performance. However, they did not show the performance of their algorithms when applied to a real-world Internet topology and did not provide results on system resource usage such as CPU cycles and memory, which are critical for a router.

The work most closely related to this project is SMALTA [18], which proposed an update handling algorithm based on ORTC and implemented it in Quagga. In [19], we have shown through simulation that our FIFA algorithms (implemented in this work) outperform SMALTA. In this work, we focused on implementing a complete system to emulate a real network that takes real-time BGP updates and archived BGP data as input. This system can be used to evaluate not only FIB aggregation algorithms but also other improvements to the Internet routing system.

Conclusion and Future Work

We have designed and implemented a flexible system that can receive real-time BGP messages, simulate any network, monitor the performance, and demonstrate the results to any individual who has access to the Internet. Using this system, we were able to test the effectiveness of our FIB aggregation algorithms quickly, and the experiment results confirm the previous findings based on our standalone simulation code.

Apart from demonstrating the performance of our FIB aggregation algorithm, there are a wide variety of applications for which this network can be utilized. First, this system can be used by network operators for internal testing prior to their actual deployment of a new feature. This framework can also be beneficial in analyzing other problems of the growing Internet. With modular changes in protocol and other data structure components, it is possible for researchers to incorporate their own algorithms in the system and study their performance at any required scale depending on their resources. Furthermore, we hope our research and development experience, especially the challenges and corresponding solutions during the implementation of FIB aggregation, sheds some light on similar problems faced by Quagga users.

We will continue to use this system to evaluate potential impact of FIB aggregation on the forwarding plane (e.g., delay and loss in data delivery). Moreover, we plan to improve the usability of our system by automating system configuration and providing an easy-to-use GUI for users to set up their experiments.

References

[1] X. Zhao *et al.*, "On the Aggregatability of Router Forwarding Tables," *Proc. IEEE INFOCOM*, 2010.
[2] Y. Liu *et al.*, "Incremental Forwarding Table Aggregation," *Proc. IEEE GLOBECOM*, 2010.
[3] Y. Liu, B. Zhang, and L. Wang, "FIFA: Fast Incremental FIB Aggregation," *Proc. IEEE INFOCOM*, 2013.
[4] "Quagga Routing Software Suite, GPL Licensed," http://www.nongnu.org/quagga/
[5] "Oracle VirtualBox," https://www.virtualbox.org/
[6] "BGPSimple," http://code.google.com/p/bgpsimple/
[7] "BGP Monitoring System (BGPmon) — Colorado State University," http://bgpmon.netsec.colostate.edu
[8] Advanced Network Technology Center and University of Oregon, "The RouteViews Project," http://www.routeviews.org/
[9] "Global BGP Table Growth," http://en.wikipedia.org/wiki/File:BGP Table growth.svg
[10] K. Fall *et al.*, "Routing Tables: Is Smaller Really Much Better?," *Proc. ACM HotNets*, 2009.
[11] "DRAMeXchange," http://www.dramexchange.com/.
[12] X. Zhao, D. J. Pacella, and J. Schiller, "Routing Scalability: An Operator's View," *IEEE JSAC*, vol. 28, no. 8, Oct. 2010, pp. 1262–70.
[13] Y. Liu, S. O. Amin, and L. Wang, "Efficient Fib Caching Using Minimal Non-Overlapping Prefixes," *SIGCOMM Comp. Commun. Rev.*, vol. 43, no. 1, pp. 14–21, Jan. 2013, http://doi.acm.org/10.1145/2427036.2427039.
[14] R. Draves *et al.*, "Constructing Optimal IP Routing Tables," *Proc. IEEE INFOCOM*, 1999.
[15] N. Sarrar *et al.*, "Leveraging Zipf's Law For Traffic Offloading," *ACM SIGCOMM Comp. Commun. Rev.*, vol. 42, no. 1, 2012, pp. 16–22.
[16] N. Sarrar *et al.*, "Exploiting Locality of Churn for Fib Aggregation," tech. Rep. 2012/12, Technische Universität Berlin.
[17] M. Bienkowski and S. Schmid, "Competitive Fib Aggregation for Independent Prefixes: Online Ski Rental on the Trie."
[18] Z. A. Uzmi *et al.*, "SMALTA: Practical and Near-Optimal FIB Aggregation," *Proc. CoNEXT*, 2011.
[19] Y. Liu, B. Zhang, and L. Wang, "FIFA: Fast Incremental FIB Aggregation," Univ. Memphis, tech. rep. CS-13-004, 2013.

Biographies

JERALD PAUL ABRAHAM (jeraldabraham@email.arizona.edu) is a Master's graduate from the Computer Science Department at the University of Arizona. He received his Bachelor's in computer science and engineering from Mahatma Gandhi University, India. His research interests are in future Internet routing architectures and artificial intelligence. He worked at the Network Research Lab during his Master's and contributed toward multiple projects in FIB aggregation and named data networking.

YAOQING LIU (liu@clarkson.edu) is an assistant professor of computer science at Clarkson University, Potsdam, New York. He received his Bachelor of Computer Science degree from Dalian Maritime University, China. He received both his Master's and Ph.D. degrees in computer science (networking) from the University of Memphis, Tennessee. His research interests include networked systems (routing, security, algorithm, measurement and protocol), software defined networking, future Internet architecture, and named data networking.

LAN WANG [SM] (lanwang@memphis.edu) is an associate professor in the Computer Science Department at the University of Memphis. She holds a B.S. degree (1997) in computer science from Peking University, China, and a Ph.D. degree (2004) in computer science from the University of California, Los Angeles (UCLA). She received an Early Career Research Award (ECRA) from the College of Arts and Sciences at the University of Memphis. Her research interests include Internet architecture, Internet routing, network security, network performance measurement, and sensor networks.

BEICHUAN ZHANG (bzhang@cs.arizona.edu) is currently an associate professor in the Computer Science Department of the University of Arizona. His research interest is in Internet routing architectures and protocols. He has been working on named data networking, green networking, network topology, and overlay multicast. He received the first Applied Networking Research Prize in 2011 from the ISOC and IRTF, and the best paper award at ICDCS in 2005. He received his Ph.D. and M.S. degrees from UCLA in 2003, and his B.Sc. from Peking University in 1995.