Measuring BGP Pass-Through Times

Anja Feldmann[1], Hongwei Kong[2], Olaf Maennel[1], and Alexander Tudor[3]

[1] Technische Universität München, Germany {anja,olafm}@net.in.tum.de
[2] Agilent Labs, Beijing, China hong-wei_kong@agilent.com
[3] Agilent Labs, Palo Alto, USA alex_tudor@agilent.com

Abstract. Fast routing convergence is a key requirement for services that rely on stringent QoS. Yet experience has shown that the standard inter-domain routing protocol, BGP4, takes, at times, more than one hour to converge. Previous work has focused on exploring if this stems from protocol interactions, timers, etc. In comparison only marginal attention has been payed to quantify the impact of individual router delays on the overall delay. Salient factors, such as CPU load, number of BGP peers, etc., may help explain unusually high delays and as a consequence BGP convergence times. This paper presents a methodology for studying the relationship between BGP pass-through times and a number of operationally important variables, along with some initial results. Our results suggest that while pass-through delays under normal conditions are rather small, under certain conditions, they can be a major contributing factor to slow convergence.

1 Introduction

Even though BGP has been studied extensively within the last few years (see Tim Griffin's Web page [1] of references as a starting point) we still do not have a good understanding on how and where signaling delays occur. Unexplainable large convergence times have been reported [2]. It is unclear whether they are attributable to protocol interactions [3], implementation idiosyncrasies [4], hardware limitations or other factors [5].

While one may attribute convergence times greater than 30 minutes to route flap damping [6] and those that are less than $2 - 3$ minutes to multiples of the MRAI timer [7] the wide variability of in-between times is puzzling [8]. An unknown component of the delay is the contribution of each router along the path of the BGP update. A detailed exploration of the delay of each router and how it relates to factors, such as CPU load, number of BGP peers, etc., can help explain these observations.

We propose to explore pass-through times of BGP updates using a black-box testing approach in a controlled environment with appropriate instrumentation. To this end, we have setup a test framework that consists of:

Device under test (DUT): a router.

Load framework: that can be used to impose a specific, pre-defined load on the DUT. In our case it consists of several PCs and an Agilent router tester.

BGP workloads can be generated by the PCs as well as the router tester. The router tester is used to generate controlled rate data traffic. Tests are repeated with both PC as well as router tester generated BGP workloads.

Instrumentation framework: that allows us to measure not just the pass-through times of BGP updates but also the load imposed by the load framework. It consists of several packet-level monitors, periodic router queries and the router tester.

The testbed has wider applicability. For example, it can be used to explore other router measures, such as line card FIB convergence.

Our methodology goes beyond RFC compliance tests and benchmarks, e.g., IETF's BMWG [9], in that we do not consider pass-through times in isolation. Rather, we investigate the correlation between BGP pass-through time and router load using a variety of stressors. BGP load is affected by such variables as (1) number of peers, (2) routing table size, and (3) BGP update rate. Non-BGP related tasks are many. We do not attempt to account for all of them individually. Instead, we impose a specific background traffic load which causes the ip_input task's CPU usage to increase. To measure the CPU load we periodically sample the cumulative route processor's load. With controlled experiments we can then use previously obtained CPU load data to infer the BGP portion of the CPU load. Our initial results are about establishing a baseline for pass-through times. We also explore some aspects of stress response. A stress situation arises when the imposed load reaches the limits of DUT.

Additional parameters that may effect pass-through time include the configured I/O queue length for BGP processes, ACLs' complexity and hit ratio, BGP policy settings such as route-maps, peer-groups, filter-lists and/or communities, as well as AS prepending, etc. These are beyond the scope of this paper.

Parameter sensitivity tests and reasonable configurations were used to reduce the otherwise many experiments, given the large number of parameter value permutations.

Our experiments show that in general the DUT handles BGP stress well, which is in line with recent findings [10]. Yet the per hop BGP processing delays can be significant. Updates are only processed every 200ms even when the MRAI timer is inactive. Activating the MRAI timer adds further delay components causing higher delays occur with increasing MRAI timer values. DUT targeted traffic, even at low data rates, drastically impacts CPU load and accordingly pass-through delays. We notice that update the rate is not nearly as significant a factor for causing delays as is the number of peers. Yet high update rates occurring concurrently on multiple peers, as is happening with after router reboots, can cause problems.

The rest of the paper is structured as follows. In Section 2 we describe our methodology for measuring pass-through times and imposing a controlled router CPU load. Next, in Section 3, we describe in detail the configuration and tools that constitute our test framework. We then describe our experiments and report their results in Section 4. Section 5 concludes our paper and comments on the practical learnings of this work as applied to current operational practice and future routing protocol design.

2 Test methodology

Three topics need detailed explanation: measuring pass-through time, separating router processing delay from MRAI timer delay, and imposing a controllable load on the DUT.

2.1 Measuring pass-through times

There are three common approaches to network measurement: passively observing regular traffic, actively evaluating injecting traffic at end points within the injecting application, and passively measuring actively injected traffic. Since we operate in a testbed the first option is not applicable. Accordingly, we use specifically designed updates as active probes together with a monitoring BGP session. The timings of the probes and the resulting response updates generated by the DUT and directed to the monitoring session are passively measured using dedicated and synchronized packet capture.

Using special updates gives us the ability to impose patterns with certain characteristics, such as ensuring that the DUT will relay it to the monitoring session. The alternative approach is to replay captured BGP update traces. While this may provide useful background traffic it suffers from several shortcomings. First the pass-through time of a router depends on the settings of several BGP specific parameters such as the value of the MRAI timer. In order to distinguish the delay due to this timer from the delay due to the router we need certain update patterns which may or may not be present in regular BGP traces. Second, not all incoming BGP updates trigger an outgoing BGP update. Therefore it is hard to tell which BGP updates are discarded or are combined into other updates. Furthermore the amount of work associated with each BGP update will vary depending on its content with respect to the router's configuration.

The simplest form of a BGP update probe pattern is comprised of a single new prefix. Since the prefix is new, the update has to be propagated to all neighbors and the monitor session, policy permitting. The drawback is that the routing table size grows ad infinitum and that the available memory becomes an unintended co-variable. The table size can be controlled via explicit or implicit withdrawals. We use implicit ones since BGP withdrawal processing differs from update processing and the results would have to be separated. Implicit withdrawals on the other hand are indistinguishable from other updates. Each time a probe update for a prefix is to be sent we randomly choose an AS path length that differs from the previous path length. This ensures that the update is propagated to the monitor session and that the quality of the path improves or deteriorates with the same probability. The BGP update rate for both probes and traces can be controlled. In summary probe patters are convenient for active measurements while replaying BGP traces is appropriate for generating a realistic load on a router.

Pass-through times are not constant. They vary with the makeup of the updates and the background load caused by other factors. Accordingly, we obtain sample values by measuring the time difference between in-bound (into the DUT) probe injections and out-bound (onto the monitoring BGP session) update propagation. To avoid time synchronization errors the packet monitors are dedicated, line rate capable, capture only cards with highly accurate, synchronized clocks. Furthermore, all other BGP sessions are terminated on the same machine, either the PC or the router tester.

Throughout this paper we use $10,000$ prefixes from the 96/8 range as probe prefixes. The central part of the experiments last for 15 minutes and our probing rate is a rather low 1 update a second. This guarantees that the TCP throughput will never be problematic. We started by using 10 probing sessions but realized that interactions with a periodic timer limited the accuracy of the estimates. The periodicity of the timer in question is $200ms$ which with 10 probes per second gave us only 2 samples per timer. To increase the rate of samples per timer to 10 we increased the number of probe sessions to 50. Each probe session uses a different random offset within the second for sending its probe. This ensures that the probing is done at exponentially spaced but fixed intervals within the second. A histogram of the resulting pass-through times is plotted in Figure 1. Interestingly, the pass-through times vary from $2.4ms$ to about $200ms$ with some ranging up to $400ms$. The average pass-through time is $101ms$. The even distribution in the range of $2ms$ to $200ms$ indicates some kind of timer. Indeed, closer inspection reveals that the router limits the update processing to 5 times a second. This should result in a theoretical upper bound on the pass-through times of $200ms$. Yet some of the updates are held back for one update processing cycle. This results in pass-through time greater than $210ms$ for 1.3% of the probes.

To determine how the pass-through time compares with the one-way packet delays we also measured the one-way delay experienced by IP packets of three typical packet sizes (64, 576, 1500 bytes) that were sent during the same experiment, see Figure 1 for the 1500 byte packets. The average delays are, $0.028ms$, $0.092ms$ and $0.205ms$, hence, significantly shorter. This shows that a BGP update is delayed 11 times longer in the best case and 500 times longer on average than a simple IP packet. While this might seem significant, the total accumulated delay, at $100ms$ a hop along a 20 hop router path, would be rather small at under 2 seconds. This indicates that we have to consider additional factors.

2.2 MRAI timer delay

The purpose of the Min-Route Advertisement Interval timer [11] (MRAI) is to limit the number of updates for each prefix/session for a peer to one every x seconds. A typical value for x is 28 seconds. If the timer fires at time $t-x$ and t then all updates received and processed within this interval of size x are batched and sent shortly after time t. Based on this observation an upper bound for the pass-through time can be derived, even when the MRAI timer is active: for each MRAI timer interval consider those probes with minimal pass-through delay. A

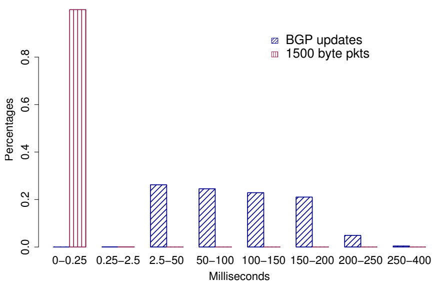

Fig. 1. Histogram of pass-through times together with one-way packet delays for typical packet sizes 64, 576, and 1500.

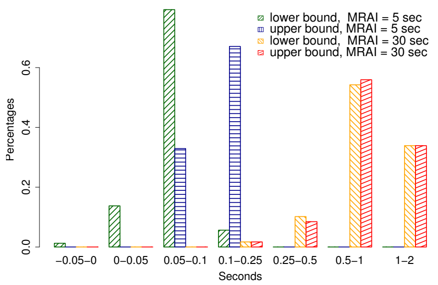

Fig. 2. Histogram of upper and lower bounds on pass-through times for MRAI values of 5 and 30 seconds.

lower bound is derivable from the probe with largest pass-through delay within the interval. This probe arrived too late to be included in the previous MRAI timer interval.

Figure 2 shows the histogram of the pass-through times for two different MRAI timer values: $5s$ and the default Cisco value, which is roughly 30 seconds. Note that with the MRAI timer in place the minimal measured pass-through times are $71ms$ and $122ms$. On the other hand the maximum values for the lower bounds are 0.121 and 1.72 seconds! This indicates that an update might have to wait a significant amount of time before it is processed even if it reaches the router at a good moment. This is especially the case for larger MRAI values where the average pass-through time increases from $109ms$ for the $5s$ MRAI timer to $883ms$ for the default MRAI value. Note that each experiment is run for the same duration. Accordingly the number of samples for the pass-through time decreases as the MRAI value increases.

Overall it seems that even for small MRAI values the timer interactions between MRAI and the BGP update processing timer increases the minimum pass-through time significantly. As the MRAI value is increased the minimum pass-through time also increases and will clearly dominate any link delays. Furthermore, inspection of the probes on the monitoring session reveals that the order of the update probes is not maintained. This means that a probe that was sent 10 seconds later than another might be observed earlier on the monitoring session.

2.3 Controlled background CPU load

Imposing a controllable background CPU load on the DUT is necessary in order to study how it responds to stress. The goal is to identify a set of tasks that generate a constant CPU load independent of BGP. This is difficult as it implies generating an input load that is uniformly served by a task running at a uniform priority. This is rarely the case. In Cisco IOS the BGP processes (and any routing tasks) have higher scheduling priority than almost everything else targeted at the CPU. The IOS ip_input task is a high priority process whose CPU use is related to the rate of a packet stream directed to the DUT's main IP address.

Another problem is measuring the CPU load. The CPU load of a Cisco router can be queried in two ways: via a command at the telnet interface of via an SNMP query. Unfortunately, the default priorities of both telnet and SNMP are lower than those of BGP and the packet processing tasks. Accordingly, for calibration only, we raised the priority of SNMP task and then measured the CPU load both via the command line as well as via SNMP for 5 rates: $2k, 5k, 10k, 15k$ pkt/s. Both estimates aligned quite well with the only problem that the command line interface did not deliver any values under high loads due to starvation. Figure 3 shows a histogram of the CPU load. $2k$ packets impose almost no load. $5k$ is already significant. $10k$ is almost critical while $15k$ is well beyond critical. Note that a $15k$ packet rate corresponds to a bit rate of 0.36 Mbits, which is rather modest for a high speed interface on a high end router. This should encourage providers to filter internal destination addresses on *all* incoming connections.

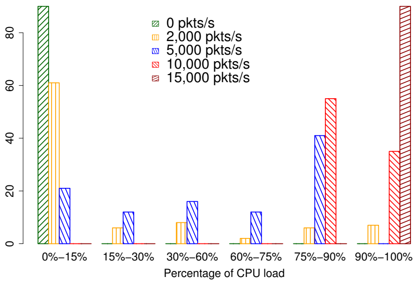

Fig. 3. Histogram of CPU load estimates for packet rates of $2k, 5k, 10k$ and $15k$ directed to the router IP.

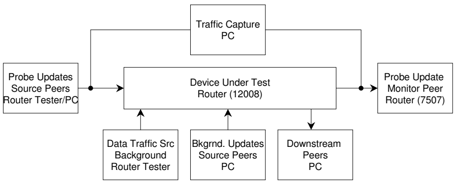
Traffic Capture PC
Probe Updates Source Peers Router Tester/PC
Device Under Test Router (12008)
Probe Update Monitor Peer Router (7507)
Data Traffic Src Background Router Tester
Bkgrnd. Updates Source Peers PC
Downstream Peers PC

Fig. 4. Test-bed setup for router testing.

3 Test framework

The testbed shown in Figure 4 illustrates its functional building blocks. The physical layout is more complex and not shown here for the sake clarity and brevity.

The device under test (DUT) is a Cisco 12008 GSR equipped with: 256MB memory, 512KB of L2 cache, 200MHz GRP CPU, three Gigabit SX and 8 Fast Ethernet interfaces. It runs IOS version 12.0(26)S. An Agilent RT900 router tester is used for selective experiment calibration and to generate data traffic.

BGP updates, probes and background, are generated by a PC. The monitoring peer runs on a Cisco 7507. Probe update traffic from the PC into the DUT is captured by Endace DAG cards [12]. Outgoing probe update traffic from the DUT to the monitoring peer is also captured by an Endace DAG card. All cards are synchronized.

The DUT is subjected to three traffic types: BGP update probes, BGP background activity updates and non-routed data traffic directed to the DUT. Probe updates are used to compute DUT pass-through times. We create a BGP activity noise floor by generating separate update streams, called background updates, that are in turn propagated to multiple downstream peers. Data traffic is used to indirectly control the CPU load and hence the time alloted to BGP processing.

DAG generated time-stamps are used to compute the DUT pass-through time. We use tethereal to decode and reconstruct the BGP TCP sessions from the capture files. To ease the configuration and setup of each experiment various scripts automatically configure the PCs, the router tester, and the routers, then start the experiments and after it is done start the evaluation. Unless specified otherwise each experiment lasts for 15 minutes actual time but the evaluation is not started for another 15 in order to retrieve all updates.

4 Pass-through times

Section 2 introduces our methodology for measuring pass-through times and shows how to impose a background load on the DUT. In this section we explore how pass-through times change as the demand on the router increases. Due to the large number of parameters we cannot test all combinations. Rather, we perform a number of tests to explore the variables to which pass-through times are sensitive, including the background CPU load, the number of sessions in combination with the BGP update rate, and the complexity of the BGP table in combination with the BGP update rate.

More precisely in a first step we combine BGP pass-through probes with the background CPU load. Next we increase the CPU load by adding 100/250 additional BGP sessions and a total of 500 BGP updates a second. This experiment uses a regular pattern of updates similar to the probes. Based on this calibration of our expectation we explore the load that is imposed by actual measured BGP tables. The next two experiments differ in that one uses small BGP tables containing between $15,000$ - $30,000$ prefixes while the other uses large BGP tables containing between $110,000$ - $130,000$ updates. Due to the memory requirements of this table the number of additional sessions is reduced to 2. This provides us with a setup to explore different BGP update rates: as fast as possible (resembles BGP session resets), 200 updates and 20 updates a second.

4.1 Pass-through times vs. background CPU load

This set of experiments is designed to show how the background CPU load influences the BGP pass-through delays. Accordingly we combine the approach

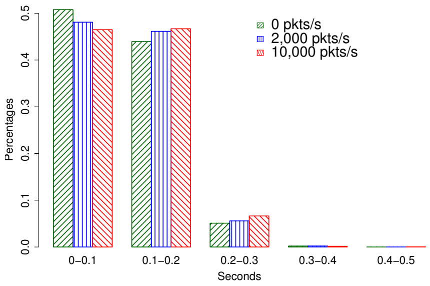

Fig. 5. Histogram of pass-through times subject to different levels of background traffic (0, 2k, 10k pkts/second).

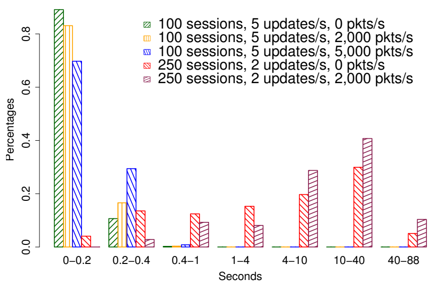

Fig. 6. Histogram of pass-through times subject to different # of sessions (100/200) and background traffic (0, 2k)).

for measuring BGP pass-through delays via active probes with that of imposing a controlled background CPU load via a controlled packet stream directed to the DUT's IP address, see Section 2. We use a packet stream of 0, 2k, and 10k packets as the first two impose no additional or just minimal load while the latter is already almost critical.

The histogram of the resulting pass-through times is shown in Figure 5. While the differences may at first appear minor the CPU load nevertheless has an impact. It causes a delayed invocation of the BGP update processing task which is reflected in the increase of the number of updates with a pass-through time larger than 210ms. With no additional load only 1.3% of the updates are in this category. With 2k packets this increases to 2.15% and for 10k packets to 3.73%. Note that a probe rate of 50 updates a second coupled with 5 invocations of the BGP update processing task every second should create delays longer than 200ms for at most 10% of the probes. Overall we conclude that the increased CPU load delays the invocation of the BGP update processing task and therefore increases the pass-through delays. Yet, due to the timer, the delay increase is on average rather small: from 101ms to 106ms to 110ms.

4.2 Pass-through times vs. number of sessions

This set of experiments is designed to explore if the number of sessions has an impact on the BGP pass-through times. So far our load mix consisted of the active BGP probes and the packets which cause a background CPU load. Next we add additional BGP sessions (100/250) and 500 updates a second to this mix. The simplest additional sessions are similar to our probe sessions with the exception that we increase the update rates to 2 and 5 updates per second respectively.

Figure 6 shows a histogram of the resulting BGP pass-through times. Interestingly adding 100 sessions poses no significant problem to the router. Yet adding 250 sessions causes way too much load on the router even without background traffic. Note that Cisco recommends to keep the number of sessions for this specific router below 150. Adding to the 100 session experiment a background CPU load of $2k$ ($5k$) increases the CPU load from a 5 minute average of roughly 67% to 83% and then to 95%. That CPU loads are not summable is an indication for the possibility of saving some CPU by delaying the processing of the BGP updates. The additional BGP sessions increase the average pass-through times to $116ms$. The CPU load is then responsible for the increase to $130ms$ and respectively $160ms$. The increase of the percentage of BGP probes that take longer than $200ms$ is even more dramatic: first from 1.3% to 7.4% and then with the packet load to 12.5% and 25.9%. Still the maximum pass-through times are reasonable small at less than $800ms$.

The further increase of the number of sessions to 250 causes a multitude of problematic effects. First the router is no longer capable of processing the updates so that it can send TCP acknowledgments on time. Accordingly the number of TCP retransmissions increases from almost none, less than 0.15%, to 2.5% of the BGP probe updates. Second the number of probes propagated to the monitoring sessions is drastically reduced. With 250 sessions the router does not propagate updates for 39.8% of the probes. This problem is aggravated (49.2%) by adding $2k$ packets of background traffic. While this reduces the number of samples of the pass-through times their values are now in a different class with average pass-through times of $9,987ms$ and $15,820ms$. The pass-through times increase by several orders of magnitude.

4.3 Pass-through times vs. BGP table size and update rate

So far all tests consisted of either probe updates or artificial patterns. Accordingly we now replace these artificial BGP sessions with actual BGP updates. For this purpose we have selected two sets of two BGP routing tables dumps from Ripe [13], one containing $37,847/15,471$ entries and the other containing $128,753/113,403$ entries. Note that routing tables generally differ in terms of their address space overlap, their size and their update rate. In this preliminary study we did not use actual updates or synthetic updates [14] with similar characteristics. Rather, since we want to study the impact of the BGP update rate on the pass-through times for different table sizes, we opted for some regular pattern. The first pattern, called "full-speed", corresponds to continuous session resets and is realized by repeatedly sending the content of the BGP table to the router as fast as possible. The second pattern, called "100 updates/sec", is similar but limits the rate of updates to 100 BGP updates a second. The third pattern, called "10 updates/sec", further reduces the rate of updates to 10 BGP updates a second. As it is well known that session resets impose a large load on the router one may expect larger pass-through times. As one hopes that the router can perform session resets at a rate of 100 updates a second the second pattern should be simpler and not impose quite such a large load. The 10 updates a second can be expected to impose even less load than our update probes and therefore should provide us with a base line.

Figure 7 and 8 show the histograms of the pass-through times for experiments with the two small tables, Figure 7, and the two larger tables, Figure 8. As expected the pass-through times for "10 updates/sec" is with an average of $111ms$ for the small table only slightly increased. The impact of the large table is visible in its average of $127ms$. For the small table the full speed update rate is significantly higher than 100 updates/sec and imposes a CPU load of 88% to 60%. This difference in terms of update rate is not as large for the full table. Here the full patter generates a CPU load of 100% as one would hope for. For the small table the average pass-through times increase significantly from $147ms$ to 181ms. If this may not seem like much there is huge danger hiding here, the one of missing BGP keep-alives. In both "100 updates/sec" experiments and the "full-speed" experiment for the large table the router did not manage to send its keep-alive in time. Therefore these experiments terminated prematurely. Overall the maximum pass-through time in the small table experiments are reasonable with a maximum of less than $710ms$ and only 35% greater than $210ms$. For the more realistic cases with the large tables this changes. Here the maximum pass-through times increase to 2.8 seconds and the percentages larger than $210ms$ increases to 76%.

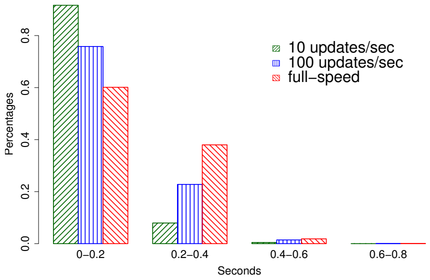

Fig. 7. Histogram of pass-through times as update rate increases (small table, 2 sessions).

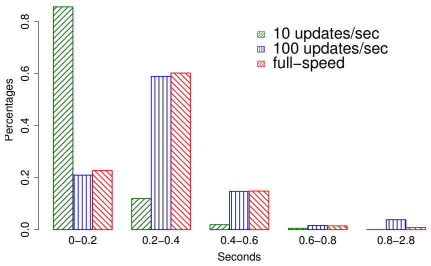

Fig. 8. Histogram of pass-through times as update rate increases (large table, 2 sessions).

5 Summary

Our results show that it is possible to determine the pass-through times using a black box testing approach. In general we found that BGP pass-through times are rather small with average delays well less than $150ms$. Yet there are situations where large BGP convergence times may not just stem from protocol related parameters, such as MRAI and route flap damping. The pass-through time plays a role as well.

Even when the MRAI timer is disabled and the router is otherwise idle, periodic BGP update processing every 200ms can add as much as 400ms to the propagation time. Increasing MRAI values appear to trigger timer interactions between the periodic processing of updates and the timer itself which causes progressively larger delays. For example, when using the default MRAI timer value even the average estimate increases to 883ms but more importantly the lower bound estimation can yield values for up to 8 seconds. This indicates that there is an additional penalty for enabling MRAI beyond the MRAI delay itself. Furthermore we have observed out of order arrival of updates which suggests that using multiple prefixes for load balancing or fail-over may not always function as expected. This bears more detailed verification.

Low packet rate data traffic targeted at the DUT can impose a critical load on the router and in extreme cases this can add several seconds to BGP processing delay. These results reinforce the importance of filtering traffic directed to the infrastructure.

As expected, increasing the update rate does have an effect on processing time, but it is not nearly as significant as adding new peers. Worth noting is that concurrent frequent updates on multiple peers may cause problems. For example, 53 peers generating 150 updates per second can cause the router to miss sending a KEEPALIVE in time, thus resulting in a session reset.

Overall the in general small pass-through times indicate that the current generation of routers may enable us to rethink some of the timer designs/artifacts in the current BGP setup. Yet care is needed to not trigger the extreme situations outlined above. Furthermore additional analysis is needed to better understand the parameters affecting BGP processing rate, such as FIB updates, line card CPU loads, BGP update contents and other configuration related parameters already mentioned above.

References

1. T. Griffin, "Interdomain Routing Links." http://www.cambridge.intel-research.net/ ~tgriffin/interdomain/.
2. C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian, "Delayed Internet routing convergence," in *Proc. ACM SIGCOMM*, 2000.
3. T. G. Griffin and G. Wilfong, "An analysis of BGP convergence properties," in *Proc. ACM SIGCOMM*, 1999.
4. C. Labovitz, "Scalability of the Internet backbone routing infrastructure," in *PhD Thesis, University of Michigan*, 1999.
5. D. Wetherall, R. Mahajan, and T. Anderson, "Understanding BGP misconfigurations," in *Proc. ACM SIGCOMM*, 2002.
6. Z. M. Mao, G. Varghese, R. Govindan, and R. Katz, "Route flap damping exacerbates Internet routing convergence," in *Proc. ACM SIGCOMM*, 2002.
7. T. Griffin and B. J. Premore, "An experimental analysis of BGP convergence time," in *Proc. International Conference on Network Protocols*, 2001.
8. Z. M. Mao, R. Bush, T. Griffin, and M. Roughan, "BGP beacons," in *Proc. Internet Measurement Conference*, 2003.

9. H. Berkowitz, E. Davies, S. Hares, P. Krishnaswamy, and M. Lepp, “Terminology for benchmarking bgp device convergence in the control plane,” 2003. Internet Draft (draft-ietf-bmwg-conterm-05.txt).
10. S. Agarwal, C.-N. Chuah, S. Bhattacharyya, and C. Diot, “Impact of BGP dynamics on router CPU utilization,” in *Proc. Passive and Active Measurement (PAM)*, 2004.
11. Y. Rekhter and T. Li, “A Border Gateway Protocol 4 (BGP-4),” 1995. RFC 1771.
12. “ENDACE measurement systems.” http://www.endace.com/.
13. RIPE’s Routing Information Service Raw Data Page. http://data.ris.ripe.net/.
14. O. Maennel and A. Feldmann, “Realistic bgp traffic for test labs,” in *Proc. ACM SIGCOMM*, 2002.