Accelerated Processing of Historical BGP Events for Testing New BGP Heuristics

Mattia Rossi, Grenville Armitage
Centre for Advanced Internet Architectures, Technical Report 090321A
Swinburne University of Technology
Melbourne, Australia
mrossi@swin.edu.au, garmitage@swin.edu.au

***Abstract*—This paper describes a technique for artificially accelerating 'real time' when testing new BGP protocol enhancements using historical real-world data. We show how months of BGP advertisement data may be processed in hours, yet generate outputs that appear to reflect months of actual operation by a network of fully featured BGP speakers. Using Quagga (an operational open-source implementation of BGP) we characterise the performance trade-offs of our technique, and show how 'accelerated time' benefits researchers who are exploring modifications to BGP's dynamic, timer-based behaviours. We consider the impact of our technique when multiple instances of Quagga run on a single host or are distributed across multiple hosts.**

Keywords: *BGP, routing, modelling, simulation, Quagga*

I. Introduction

Border Gateway Protocol (BGP) version 4 is central to the internet's inter-domain routing infrastructure [1]. BGP is spoken by border routers between autonomous systems (AS), sharing reachability information with neighbors in order to build and maintain up-to-date maps of AS and route topology. These maps are constantly in a state of flux due to changing policies between different organisations, link failures and restorations, or attempts to locally optimise the distribution of traffic between adjacent AS. BGP is essentially a continuous, distributed computation based on incomplete information shared by neighbors. Many activities (such as advertising and withdrawing routes learned from other neighbors) are controlled by local per-border-router timers that cause complex network-wide consequences.

Researchers exploring optimisations of BGP's behaviour must test not only for functional correctness but for viability. By viability we mean that the modified behaviour does not have unforseen and/or negative consequences when exposed to long-term patterns of realistic BGP activity. However, testing for long-term viability is currently a significant challenge.

One approach is to simulate a network of BGP speakers fed by streams of synthetic BGP updates. Whilst it is possible to operate a simulation at much faster than real time, the results still need verification in operational networks. The simulator's BGP code base is unlikely to have actually been debugged in real networks, and the synthesised BGP updates are only statistical approximations of the diversity of update traffic seen in real networks.

A second approach is to implement and test new ideas in one or more BGP speakers that are receiving a live feed of BGP updates from a real network. Key limitations are that events happen at the speed of the real world (a week of activity takes a week to observe) and tests cannot be repeated with identical input data. One might also drive the modified BGP speakers with a historical feed of BGP updates previously collected from a real network. This has the advantage of being repeatable (the historical data may be replayed many times to assess permutations of a modified BGP algorithm). However, observing realistic timer-related BGP behaviours requires the historical updates be regenerated at their original arrival rate. A month of historical data would take a month to use.

This paper describes and evaluates a variation on the second approach. We enable one or more real BGP speakers to operate in an accelerated time frame synchronised to the timestamps of update messages previously collected from a live network. For example, a week of historical BGP updates may be evaluated in hours yet the BGP speakers will produce outputs correctly scaled in time.

Our accelerated emulator is built around Quagga [2], an operationally-tested, open-source BGP router implementation. Multiple instances of Quagga can be used to emulate arbitrary BGP topologies operating in accelerated time. The instances of Quagga may exist on a single host or be distributed among a small number of hosts. The accelerated emulator has become a valuable tool for our evaluation of different techniques for path exploration damping using weeks and months of real-world BGP update data [3] [4] in Quagga.

Section II discusses the background for this work, with our basic technique described in section III. Our analysis of resource consumption when emulating different small BGP topologies is provided in section IV. Section V considers possible future work, and the paper concludes with Section VI.

II. Background

Our primary motivation was to enable research into new heuristics for processing BGP advertisements, which required that we repeatedly play-back historical data gathered at various points in the internet. A number of techniques were considered before settling on Quagga for our accelerated emulation system.

Arbitrary BGP topologies can be simulated in an accelerated fashion using BGP++ [5], a BGP plug-in for the NS-2 [6] simulator. BGP++ is itself based on Zebra [7], Quagga's predecessor. Due to its Zebra origins, it should be relatively easy to port BGP++ modifications into Quagga for operational use. However, as noted in section I, viability testing of new BGP algorithms would be still rely on synthesised approximations of real-world BGP update traffic patterns.

An alternative is BGPMon [8], designed to process BGP updates from either a live BGP network or historical real-world data. BGPMon is is primarily intended to provide real BGP peering sessions for research, and lacks the ability to install learned BGP routes into an underlying system's forwarding tables. BGPMon currently does not provide any mechanism for accelerated processing of historical BGP update streams.

We ultimately chose Quagga because it is an open-source routing suite with fully featured BGP implementation. Quagga is considered to be conservative in resource usage and has been deployed for operational use in various AS. Our technique for accelerated processing was relatively easy to back-port into Quagga's codebase, and our sources of historical BGP update data were in a Quagga-compatible binary MRT [9] form. Implementing and testing new ideas in Quagga would provide the best outcome for a BGP research project, as the resulting code could be directly used in an operational environment.

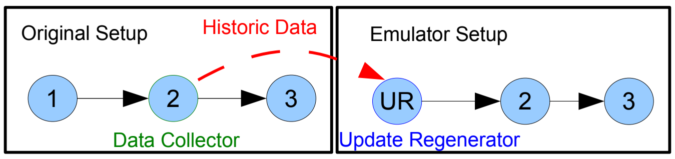

Fig. 1. BGP advertisements originally collected at router 2 can be later replayed by our Update Regenerator (UR) as though they were again originating from router 1

III. Emulator Setup

Our technique involves two steps - regenerating authentic BGP update messages from historical data and accelerating the time frame in which the regenerated stream of updates is processed.

A. *Regenerating historical BGP update traffic*

Figure 1 illustrates the relationship between the source of historical data and our later regeneration. In the Original Setup router 2 collects BGP advertisements received from adjacent router 1. Node 1 is typically identified as the source of each advertisement, with the AS number and IP address of the adjacent router coded in the nexthop and AS path attribute of the BGP packets collected by node 2. Consequently, in the Emulator Setup our *Update Regenerator* (UR) uses router 2's logfiles to recreate a BGP session that appears to be coming from router 1

We implemented our UR to read logfiles of BGP update activity stored as text form or MRT dump files[1]. The UR establishes a legitimate BGP session to a live instance of Quagga (router 2, in Fig. 1) and regenerates previously saved BGP advertisements at an accelerated rate[2].

B. *Accelerating time*

Our goal is to accelerate the emulation of BGP topologies having nodes represented by multiple instances of Quagga. Figure 2 expands on Figure 1 to illustrate our solution.

In the original setup a real BGP speaker feeds updates to Quagga 1 (router 2), which processes and forwards updates to Quagga 2 (router 3) as required. Quagga's core clock normally retrieves UTC timestamps from the

[1]When used to collect traffic, Quagga may log BGP advertisments in either human readable text form or binary MRT dump files

[2]Our prototype UR uses the local loopback interface to open a BGP connection to this 'router 2' instance of Quagga running on the same host. 'Router 2' may then have any arbitrary AS number and IP address

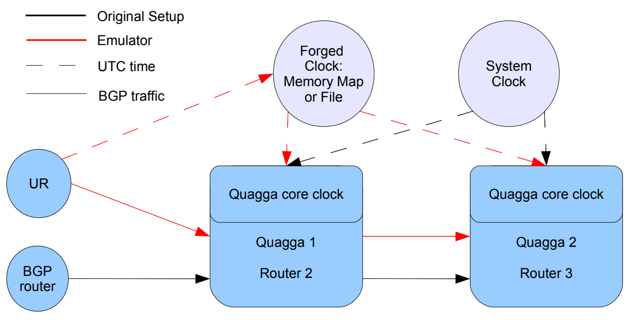

Fig. 2. BGP updates are triggered by time based events. (black lines) Original setup: Quagga's core clock is derived from the system clock when processing real updates. (red lines) Emulator setup: The UR establishes a 'forged clock' in shared file or memory map. Quagga's core clock now reads this file or memory map instead.

host's system clock in order to drive Quagga's event system responsible for triggering BGP updates.

In our accelerated emulator setup we modify each participating Quagga to instead read their core clock's timestamps from a block of shared memory (or file) controlled by the UR. The UR itself increments this alternative time information using the timestamps of re-generated BGP advertisements, effectively 'accelerating time' for all instances of Quagga participating in the emulation[3].

Our approach provides the significant benefit of reducing the time it takes to evaluate new BGP algorithms using historical BGP data. However, there are resource consumption and performance issues to be considered.

Using mapped shared memory restricts our emulator to a single host or system where all processes can access the same shared memory. Using a file enables us to run all Quagga instances on a single host or to distribute the load over a number of hosts (using a network file system to provide all processes with access to the timestamp file). The last approach would ease the load on a single system, but would dramatically increase the network load. In the following section we consider resource consumption for these scenarios.

IV. Resource Usage and Limitations

In this section we are presenting some facts about resource usage of the accelerated emulator. We have tested three different approaches, regarding their way of exchanging time information:

[3]Quagga's events system is also changed to solely utilise information in the timestamps rather than on the frequency of timestamp update

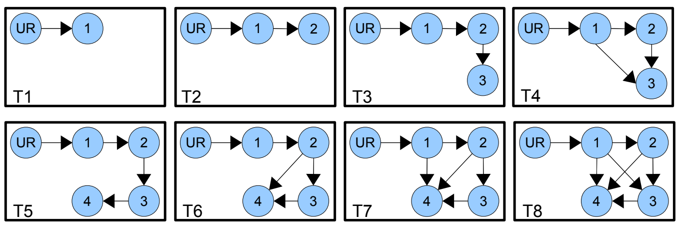

Fig. 3. Various topologies have been tested for resource usage tests. UR is the Update Regenerator which reproduces the historic data stream. The arrows indicate the direction of BGP advertisement propagation

- The UR and all Quagga instances running on the same host using a memory map for the fake clock
- The UR and all Quagga instances running on the same host using a file for the fake clock
- The UR and at least one Quagga instance running on a single host, connected to at least one other Quagga instance on an other machine over a 100Mbit single switched ethernet connection, using a file shared via NFS [10] for the fake clock. The network file system uses the same ethernet connection.

We have tested also various topologies of routing networks, in order to be able to make a comparison of resource usage as shown in Fig. 3.

All tests have been carried out on hosts equipped with a dual core 2.4 GHz processor and 4GB of RAM running on FreeBSD Current (8.0) in 32 bit mode. All tests have also made use of a MRT dump file which propagates an initial routing table followed by BGP advertisements. The file contains 2 hours worth of real BGP traffic collected on a router running Quagga 0.99.10, which connected to a single router (AS 65000) at APNIC (Asia Pacific network Information Centre) [11]. The routers used in the tests were Quagga 0.99.10 based routers with a slight modification for enabling use of the fake time, the Update Regenerator was written in Python.

A. *Memory Mapped Emulator*

Analysing the results of the memory mapped emulator over topologies T7 and T8 of Fig. 3, we can see that CPU usage, Memory usage and with that the overall execution time already rise with one additional connection as depicted in Fig. 4.

If we look at the CPU and memory usage of the UR, we see that it uses always the same amount of resources. Only the overall time changes, which depends on the overall system load, which again is defined by

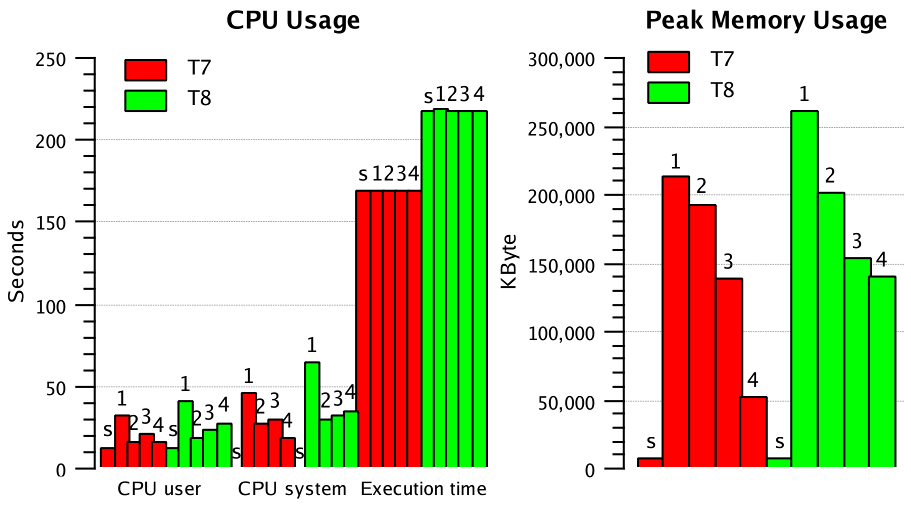

Fig. 4. Execution time, CPU user time, CPU system time and peak memory usage of the mmap version emulating topologies T7 and T8

the amount of Quagga instances and peering sessions. Additional peering sessions between Quagga instances are mostly responsible for extra load on the system. The extra load is added to the routers participating at the peering session (routers 1 and 4). As the emulator is not deterministic in its behaviour[4] the little extra load on routers 2 and 3 can be ignored.

Additional peering sessions put different load on the system, depending on the peers involved. A peer generating high load will generate more additional load with one additional connection, than a peer with less load. Fig. 5 compares peak memory and average CPU usage of topologies T1 to T8, which reflect such situations. The graph contains a comparison to the resource usage of "original" Quagga instances, which make use of the real system clock.

The higher memory usage of T4 towards T6 is caused by the additional connection starting from router 1 in T4 instead of router 2 as in T6. As router 1 is usually on a higher load as can be seen in Fig. 4, the additional peering session adds much more load to the system as an additional peering session from router 2 with an additional Quagga instance. T1 and T8 use slightly more memory than the comparable original Quagga instances. This amount equals approximately the amount used by the Update Regenerator, which is not taken into consideration when using original Quagga instances, as they connect to a live source. CPU usage is scaled to 200 percent as the host is equipped with a dual-core CPU. We can see that the emulator uses much more CPU than Quagga instances which run at normal speed. Topology T4 again uses slightly more CPU than T6 for the same reason as above. Additionally we see a slight decrease in CPU usage for T8 towards T7 which is caused by linear increase of execution time and CPU time resulting in constant percentage. This indicates that in this test runs we might have already reached the limits of the host.

[4]Quagga instances run completely independent, therefore slight variations in event handling may occur between test runs, resulting in slightly different output and resource usage

B. File Based Emulator

A file based emulator is much slower than a memory map based one in processing the same BGP data. The continuous file operation adds extra load to the system, and CPU usage as well as overall execution time increase as memory usage does too. Fig. 6 compares memory map based and file based resource usage. As for the memory mapped emulator, it can be also seen, that the replaying script uses a quite constant amount of CPU and a minimal amount of memory.

C. File Based Distributed Emulator

The distributed file based emulator is quite different to the systems described before. As it can make use of multiple networked machines, it has a much lower resource usage per machine. It uses a network file system to propagate the time information, therefore it generates additional network traffic. We analysed specifically topology T8 in this system, with 3 setups:

- 3 Quaggas on the NFS server, 1 on the client (3to1)
- 1 Quagga on the NFS server and 3 on the client (1to3)
- 2 Quaggas on both (2to2)

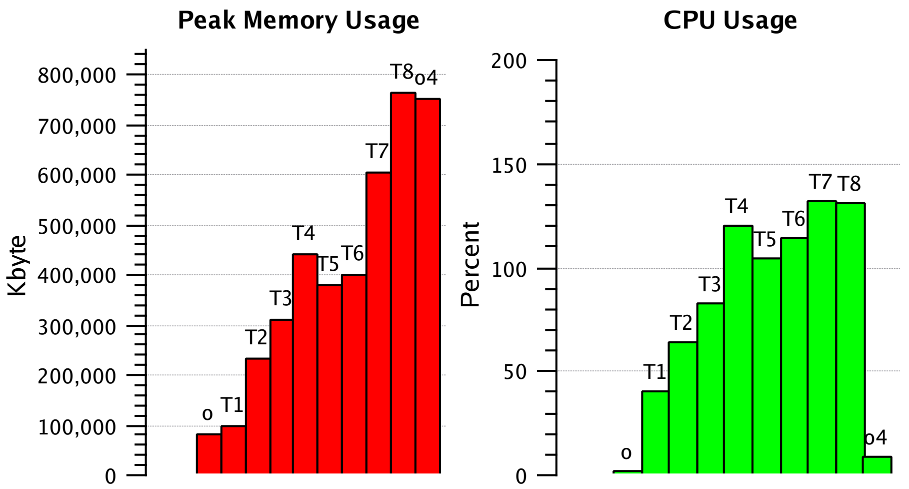

Fig. 5. Memory and CPU usage for topologies T1 to T8 on the memory map emulator. It also compares to a single original Quagga instance (o), and a T8 like topology of 4 full mesh Quaggas (o4).

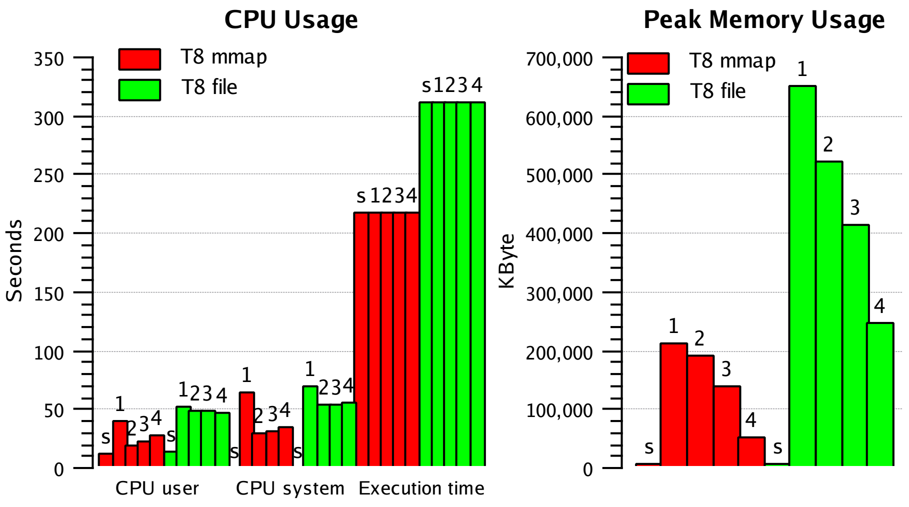

Fig. 6. Resource usage for topology T8 using a memory map based emulator vs. a file based emulator

Figure 7 shows the network traffic of all 3 settings over the single Ethernet link connecting the two machines. It compares BGP and NFS traffic and also shows the typical BGP traffic that is produced over a single peering session between two original Quagga instances as well as the real network traffic generated by the non distributed emulators (which equals zero as traffic is sent over the loopback interface).

From the graph it can be seen that the average NFS traffic overhead is remarkable compared to the generated BGP traffic. If we consider the maximum average network usage of 857 KByte/s (approx. 6.8 Mbit/s) achieved in this setup, we see that with the distributed file based emulator we would be able to emulate larger topologies than with an mmap based one, not having reached common network limits by far. Additionally it can be seen, that NFS overhead increases only about approx. 150 KByte/s for two additional Quagga instances accessing the NFS on the same client (“1to3” compared to “3to1”), which shows that the additional overhead per BGP speaker instance and NFS client is growing only slowly. The increase in BGP traffic for the “3to1” setting shows, that BGP traffic increases towards the sink of the BGP network.

Sharing time information through the NFS file system though, results in a less accurate output. Running the distributed file based emulator at its maximum speed, we could identify skipped timestamps in the shared file used by the Quagga instances on the NFS client host compared to the timestamps written at the NFS server

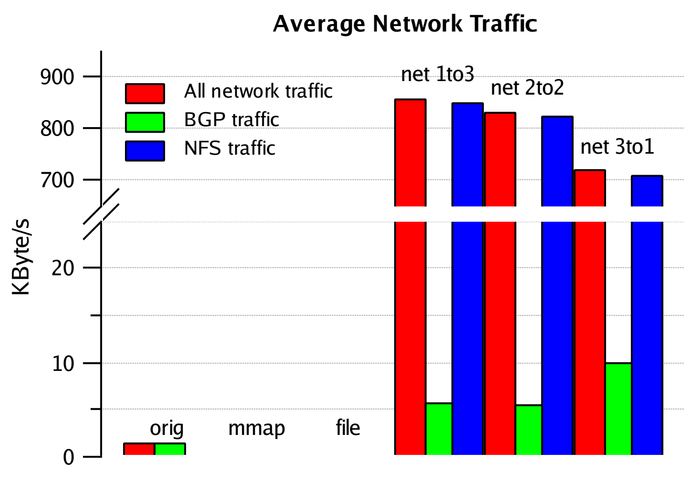

Fig. 7. Average network overhead of the different emulator types.

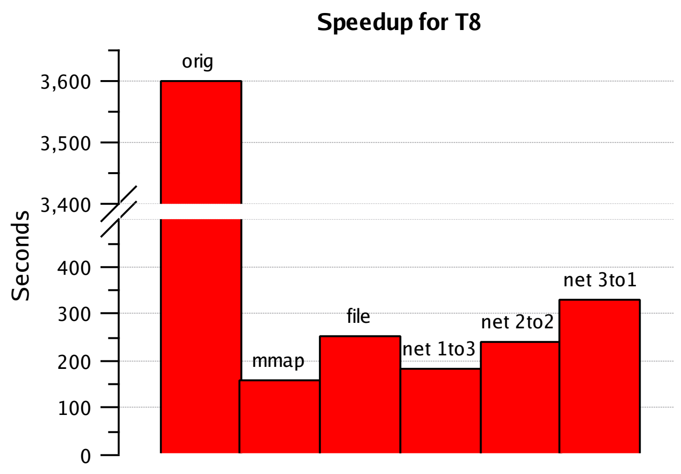

Fig. 8. Comparison of the time needed to process 2 hours of BGP data over topology T8 with the various types of emulator settings

side. The accuracy could be improved by artificially slowing down the propagation of BGP advertisements in the UR, but as the NFS based emulator has been used only for demonstration purposes, it has not been further analysed.

Figure 8 shows how much speedup could be achieved for every type of emulator over topology T8. The comparison is towards an original BGP session lasting 2 hours.

It can be seen that a file based emulator running many Quagga instances and peering sessions on the same machine as the UR, has the lowest speedup ("file", "net 3to1"). The best performance is achieved with the emulator using a memory map, which reduces the processing time from the original 2 hours down to 2 minutes and 37 seconds resulting in a speedup of approximately 24.

V. Further Work

In this project we used Python to create the Update Regenerator and Quagga as the routing software of choice as described in II. The technique presented although may be applied to a large variety of BGP routing software, using other programming languages than Python for the UR and developing new ways to provide fake time to the participating routers. This could result in performance improvement and allow the emulation of larger BGP topologies. Creating an emulator with BGP speakers distributed over different hosts using faster network file system protocols than NFS, would also improve accuracy allowing to operate still at higher speed.

VI. Conclusions

By extracting timestamp information about BGP update events stored in a historical BGP data stream, and providing them to Quagga based BGP speakers to forge the synchronisation with the system clock, we were able to replay this historical data stream accelerated over a system of these Quagga instances, still producing outputs correctly scaled in time. We have been able to emulate various BGP topologies using three different approaches to share fake time information between the participating BGP speakers, and we have also shown possible resource usage.

We identified the memory map based emulator as the fastest approach with a speedup of 24 times using a full mesh topology over four Quagga instances, meaning that 24 hours of data could be processed in one hour. Smaller topologies with e.g. three Quagga instances and two peering sessions, could be sped up to 71 times, allowing the processing of one months data in only 10 hours. The size of the emulated BGP topology is limited to the available resources on a single machine, which might be exhausted with only few Quagga instances and peering sessions. This approach is suggested for testing new Quagga implementations on small topologies with historical data covering large periods of time.

The distributed file based approach was used for demonstration purposes only, and showed a slightly smaller speed up (about 20 times) for the full mesh BGP topology, but required less system resource usage being distributed over several machines and having the trade off as extra load on the network. Additionally the accuracy of the output was deteriorated, resulting in an even lower speedup, if we wanted to keep the accuracy high. We could also show that in our test case the distributed file based approach would allow the emulation of larger topologies than a memory map based one. This approach is suggested for emulating larger topologies with historical data covering small periods of time.

Based on this facts we are able to confirm the benefits such an emulator can provide to BGP research: it makes use of a full featured BGP routing software (Quagga), which is ready for use in operational networks, it makes use of real historical BGP data, allowing to repeat tests with the same input keeping output coherent, and the system can process data faster than real-time, showing that it's speedup mainly depends on the size of BGP topologies simulated.

VII. Acknowledgements

This research project has been made possible in part by a grant from the Cisco University Research Program Fund at Community Foundation Silicon Valley.

References

[1] Y. Rekhter, T. Li, and S. H. (Editors), "RFC 4271: A Border Gateway Protocol 4 (BGP-4)," RFC 4271 (Draft Standard), 2006, obsoletes RFC 1771. [Online]. Available: http://tools.ietf.org/html/rfc4271

[2] K. Ishiguro, "Quagga Software Routing Suite." [Online]. Available: http://www.quagga.net

[3] G. Huston, "ISP column: Path Damping," June 2007. [Online]. Available: http://www.potaroo.net/ispcol/2007-06/dampbgp.html

[4] M. Rossi, "Implementing path-exploration damping in the Quagga Software Routing Suite Version 0.99.10," Centre for Advanced Internet Architectures (CAIA) - Swinburne University of Technology, Tech. Rep., November 2008. [Online]. Available: http://caia.swin.edu.au/reports/081117A/CAIA-TR-081117A.pdf

[5] X. Dimitropoulos and G. Riley, "Efficient large-scale BGP simulations," *Elsevier Computer Networks, Special Issue on Network Modeling and Simulation*, vol. 50, no. 12, 2006.

[6] DARPA, "The Network Simulator NS-2." [Online]. Available: http://www.isi.edu/nsnam/

[7] K. Ishiguro, "GNU Zebra: Free routing software distributed under GNU General Public License." [Online]. Available: http://www.zebra.org

[8] D. Massey, D. Matthews, H. Yan, and Y. Chen, "BGPMon: BGP Monitoring System." [Online]. Available: http://bgpmon.netsec.colostate.edu

[9] L. Blunk, M. Karir, and C. Labovitz, "MRT Routing Information Export Format," June 2008. [Online]. Available: http://tools.ietf.org/html/draft-ietf-grow-mrt-08

[10] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck, "RFC 3530: NFS version 4 Protocol," RFC 3530 (Draft Standard), 2000, obsoletes RFC 1813, RFC 1094. [Online]. Available: http://tools.ietf.org/html/rfc3530

[11] G. Huston, "AS65000 BGP Routing Table Analysis Report." [Online]. Available: http://bgp.potaroo.net/as2.0/index.html