Creating Realistic BGP Models *

Xenofontas A. Dimitropoulos
George F. Riley
College of Engineering
Department of ECE
Georgia Institute of Technology Atlanta, GA 30332-0250
{fontas,riley}@ece.gatech.edu

Abstract

Modeling the Internet infrastructure is a challenging endeavor. Complex interactions between protocols, increasing traffic volumes and the irregular structure of the Internet lead to demanding requirements for the simulation developer. These requirements include implementation detail, memory efficiency and scalability, among others. We introduce a simulation model of the Border Gateway Protocol that we call BGP++, which is built on the popular ns-2 *simulation environment. A novel development approach is presented that incorporates the public domain routing software GNU Zebra in the simulator. Most of the original software functionality is retained, while the transition to the simulation environment required a manageable amount of effort. Moreover, the discussed design inherits much of the maturity of the original software, since the later is only minimally modified. We analyze BGP++ features and highlight its potential to provide significant aid in BGP research and modeling.*

1. Introduction

BGP [15] is the only inter-domain routing protocol of the Internet, deployed for more than a decade. BGP's role is to maintain reachability among the autonomous systems (ASs) that comprise the Internet. Reachability is maintained under a set of restrictions imposed by applied policies. BGP policies are determined by commercial agreements between ASs and are applied mainly in the form of filters in the incoming and outgoing interfaces of the routers. The protocol conveys information for networks spread in adverse geographical locations bringing them together in the Internet.

In the recent years, BGP has received a significant amount of research interest. Attributes studied are its stability [7, 19], convergence time [11, 12], path inflation [6, 18], policy atoms [1, 2] and path richness [17]. Works of researchers reveal several problems, the majority of which are related to BGP dynamics and rise from BGP's complexity. The distributed nature of the algorithm, the flexibility left to the operator in the form of policies, and the scale of the Internet exacerbate BGP abnormalities and make the study of its behavior very difficult. The methods of choice in recent studies are passive measurements, active measurements, analytical modeling and simulations. Our focus is on the development of BGP simulation models. Accurate simulation models are essential to understanding the inter-domain routing infrastructure, to recreate observed patterns, to predict future behavior and to examine the applicability of proposed modifications. The only other detailed BGP simulation model to our knowledge is built on SSFnet simulator [3, 4] and is used in [8] to examine interdomain routing convergence time.

Our work introduces an implementation of BGP protocol for the *ns-2* [13] simulator that we call BGP++. In contrast to conventional simulation software development techniques, BGP++ is built from existing software. In particular, GNU Zebra BGP daemon (*bgpd*) [10] is modified to work in a simulation environment. The advantage of this approach is that the simulator inherits much of the detail and the functionality of the original software, while moderate development effort is required.

Zebra is an open source implementation of BGP as well as other routing protocols. It has been used by network operators as part of the Internet infrastructure. Furthermore, *ns-2* is a widely used discrete event simulator that provides implementations of many networking protocols. The combination of the two software packages leads to detailed models of BGP.

The remainder of this paper is organized as follows: Sec-

---

tion 2 describes in detail our method in incorporating Zebra in *ns-2*. Section 3 describes BGP++ features, validation tests and performance measurements. Section 4 introduces the parallel and distributed version of the simulator. Finally, section 5 gives some conclusions and future directions in our research.

2. Integration Methodology

Throughout the research community simulations are used extensively to examine the behavior of various network protocols and architectures. Because of their importance, their development has become part of the research. Common characteristics sought are implementation accuracy, memory efficiency, and scalability. Public domain software often provides accurate and efficient implementations of protocols. In this section we analyze how the public domain Zebra software was modified to work with *ns-2* simulator.
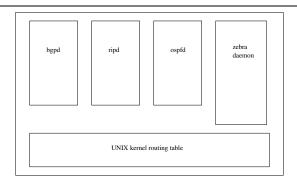

Figure 1. Zebra software modular architecture

Initially, Zebra routing software is examined to determine whether its software architecture is suited to the desired goal. Zebra is written in C and implements other routing protocols as well as BGP. It has a separate daemon for each routing protocol implemented that can be run as a stand-alone process. Another daemon, as shown in Figure 1, takes care of communications between routing daemons and the kernel routing table or other routing daemons. This scheme provides a modular architecture with independent, well-separated implementations for each daemon. As a result, the BGP implementation is easily identified and extracted from the software. Zebra's *bgpd* implementation and library methods form the basis on which the BGP++ simulator is built. The logic of the code that implements BGP algorithms is not changed. However, features intrinsic to operating systems are removed or modified to meet the characteristics of the simulator. The typical public-domain software that implements networking protocols for open source

operating systems is written in C, uses the BSD socket API, makes use of system calls, and has at least one blocking routine. On the other hand, *ns-2* is written in C++ and follows the discrete-event simulation paradigm. The following list identifies differences between discrete-event simulation software and corresponding open-source platform software that are explored during the development of BGP++:

- Use of C++ compared to C
- Use of discrete event scheduling compared to process-based scheduling algorithms
- Use of simulator TCP implementation compared to BSD sockets
- Use of nonblocking routines compared to blocking routines

The rest of this section elaborates on the implementation of BGP++. A fundamental requirement for the simulator is to initiate multiple BGP daemons. In contrast, Zebra software is design to run one *bgpd* per process. To accommodate this requirement we convert Zebra's C code to C++. This is done by creating a class named BGP that contains most of the C code. All the C functions are turned into C++ member functions and all global variables are turned into member variables. More sophisticated schemes for converting C code into C++ can be developed that take into account the semantics of the C code.Our approach considers that the different parts of the Zebra's C code are correlated in that they constitute a *bgpd*. Thus, they can be incorporated into a single BGP class.

Interleaving Zebra *bgpd* scheduler with *ns-2* scheduler is the second requirement. Discrete event simulators use queue based schedulers. The entries of the queue are events that are sorted in a time-stamp order. On the other hand, system software has no typical scheduling mechanism. Scheduling varies with developer's design from simple to arbitrarily complex. The typical networking application executes code until a blocking routine is reached, at which point the application blocks. To incorporate the application software in the simulator, the application scheduler has to be modified to communicate with the queue scheduler. A high-level model of the way the interleaved scheduling works is as follows: whenever there is an event for the application, e.g., a start event, the simulator gives control to the application to execute the associated code. The application continues until the first blocking routine is fetched; then, instead of blocking, it returns control to the scheduler. The application is given control again when the blocking routine would unblock. Events that could unblock the application are read events, e.g. a packet arrival; write events, e.g. a buffer becomes writable; timer expirations and user triggered events.

Zebra scheduling is based on the *select()* system call. *select()* takes as arguments a list of file descriptors and a timeout value. It blocks until a file descriptor changes status or until the timeout expires. The file descriptors indicate I/O streams, while the timeout value is set to the next timer expiration time. If *select()* unblocks, the appropriate code is called. Execution continues until *select()* is reached again through an infinite loop.

The interleaved scheduling works as follows: at the start event *ns-2* calls the simulated BGP daemon to make required initializations. Those are a subset of the initialization in the *main()* function of Zebra's *bgpd*. The BGP finite state machine is entered, and execution proceeds until the blocking routine *select()* is reached. When this happens, *select()* is not executed, instead an event for the calculated timeout value is entered in the *ns-2* scheduler before control is returned to the simulator. If we disregard events that unblock the simulated BGP daemon, the later will take control as soon as the timeout expires. However, *select()* could unblock before the timeout expires. For instance, Zebra *bgpd select()* unblocks when there is a read or write event [1]. For this reason, the simulator has to invoke the simulated BGP daemon upon a read or write event. In operating systems, read events occur when the TCP stream has new bytes available. In the simulated BGP daemon, upon a packet arrival, *ns-2* cancels the future timeout event for this daemon and gives control to daemon. Packet SDUs are handed to the BGP daemon by the simulator's underlying TCP implementation. In Zebra software, write events result from the fact that non-blocking output routines, namely *write()* and *writev()*, do not copy the application buffer to the kernel output buffer immediately. Instead of this, if the kernel output buffer is full, the copy operation is postponed until the buffer becomes writable, i.e. write event. This time interval is usually very small and is ignored in our BGP models.

A common simplification in network protocols modeling is the omission of the CPU processing time. This is a valid assumption as long as the processing time is very small. BGP routers can exhibit long processing times especially when their routing tables are big. Typical routing tables of core Internet BGP routers have more than 100,000 entries. For this reason, we implement a workload model that adds delay representing the finite execution time. The way the workload is modeled is that when a simulated BGP daemon completes an operation, like parsing a packet that just arrived, it picks a *busy-period* time value. This time value represents the finite execution time of the operation that just completed. The BGP daemon waits for *busy-period* time before executing any following operation. Also, during this period it does not respond to any invocations, e.g. packet arrivals. Instead of this, all invocations are buffered and executed in a FIFO basis as soon as all pending operations are completed. Each of the buffered invocations results in new operations and *busy-period* values. BGP++ has two workload models that differ in the way they choose the *busy-period* value. In the *uniform* workload model, the *busy-period* is a uniform random variable within a user specified range. In the *time-sample* model, the *busy-period* is calculated as the product of the CPU clock frequency and the execution cycles count. The execution cycles count is the number of CPU cycles a operation consumed. The kernel patch described in [14] provides the required utilities to monitor the execution cycles count.

The third step is to substitute the BSD socket API with the corresponding TCP implementation that is provided by the simulator. For BGP++, it is chosen to use the *ns-2 FullTcp* implementation. However, *FullTcp* does not support all features of BSD sockets. Thus, *ns-2 FullTcp* implementation is extended to notify the application as soon as the connection moves from *SYN_RCVD* or *SYN_SENT* to *ESTABLISHED* and from *ESTABLISHED* to *CLOSE_WAIT*. The first two transitions are used in modeling of BSD sockets non-blocking *connect()* and *accept()*, respectively. In both cases they notify the application that the three way hand-shake was successful. The third transition notifies the application upon passive connection termination.

An important difference between system software and simulation software is that the later may have multiple instances of the implemented protocol running in the same process, while the former facilitates only one instance per process. C++ gives the programmer the capability to instantiate multiple BGP daemons. However, since the original software is not developed in C++, some of the provided utilities are modified to accommodate the multiple-instance requirements. In particular, Zebra configuration and logging utilities are modified to support more than one configuration and logging files, respectively, per process.

The last step to introduce Zebra routing software in *ns-2* simulation environment is to replace system calls with corresponding simulator functions and remove code associated with unnecessary functions. For instance, *bgpd* supports a telnet interface that is used to configure or query the routing daemon at run-time. This facility is removed; it is replaced by a similar facility that provides the simulator user the capability to query or reconfigure the simulated routing daemons at run-time.

Implementations of simulation software are often predated by corresponding system implementations. In this cases the utilization of the later in the development of the former should be considered. Despite the fact that much of the aforementioned implementation details are specific to *ns-2* and Zebra software, our experiences can provide use-
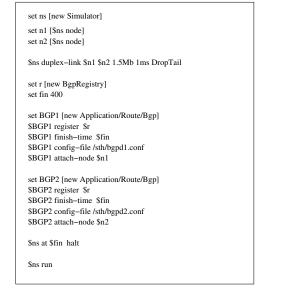
---

1   User interrupts are treated as read events, since user communication is done through a telnet interface

```
set ns [new Simulator]
set n1 [$ns node]
set n2 [$ns node]

$ns duplex-link $n1 $n2 1.5Mb 1ms DropTail

set r [new BgpRegistry]
set fin 400

set BGP1 [new Application/Route/Bgp]
$BGP1 register  $r
$BGP1 finish-time  $fin
$BGP1 config-file /sth/bgpd1.conf
$BGP1 attach-node $n1

set BGP2 [new Application/Route/Bgp]
$BGP2 register  $r
$BGP2 finish-time  $fin
$BGP2 config-file /sth/bgpd2.conf
$BGP2 attach-node $n2

$ns at $fin  halt

$ns run
```

Figure 2. Sample *tcl script that creates two BGP daemons*

```
!Local AS and local IP address
router bgp 1
bgp router-id 192.38.14.1

!Neighbors
neighbor 192.38.14.2   remote-as 2

!Local networks
network  190.0.0.0   mask 255.0.0.0
network  189.0.0.0   mask 255.0.0.0

!Enable debugging
debug bgp
debug bgp fsm
debug bgp keepalives
debug bgp filters
debug bgp events
debug bgp updates

dump bgp all dump1.log
log file bgpd1.log
```

Figure 3. Sample BGP++ router configuration file

ful guidelines in similar projects, since the high-level problems encountered remain the same.

3. The Simulator

3.1. BGP++ configuration

This section gives a high level overview of BGP++ configuration and introduces some of its utilities.

The configuration of BGP++ is a two step process. First, the simulated topology has to be configured with a *ns-2 tcl* script. Then, each simulated router is configured using a distinct configuration file. The *tcl* script specifies a topology and associates BGP daemons with nodes. A sample *tcl* configuration script is shown in Figure 2. To create a BGP router it is necessary to instantiate a BGP daemon, attach it to a node, assign a configuration file, register the daemon and set the finish time of the simulation. The registration is the process of inserting an entry in a global table that maps BGP instances to IP addresses. This table is called BGP registry and is required because BGP++ uses a dual addressing scheme, i.e. BGP routers are identified by both actual IP addresses and *ns-2* addresses. Nevertheless, *ns-2* addressing is invisible to the user. The second step of the configuration is to setup the simulated routers. Each router setup is done in a separate file using the same syntax used to configure Zebra *bgpd*. The configuration commands, such as those in the sample configuration in Figure 3, or slight variations thereof are also used by many commercial BGP implementations. The following list identifies supported RFCs.
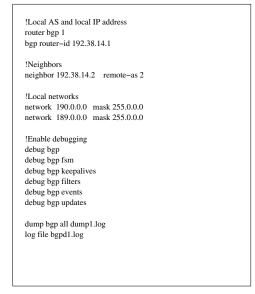
- RFC 1771 A Border Gateway Protocol 4
- RFC 1965 Autonomous System Confederations for BGP
- RFC 1997 BGP Communities Attribute
- RFC 2796 BGP Route Reflection
- RFC 2918 Route Refresh Capability for BGP-4

Finally, BGP++ provides support for run time commands. The user can query the simulated routers using the *show* command variants or can change the configuration at run time. A *tcl* command provides this functionality. The *tcl* command takes as arguments the BGP configuration command to be called, the calling time and the *bgpd* by which the command should be executed. This utility replaces the functionality available through Zebra telnet's interface.

3.2. BGP++ validation and verification

In this section we discuss BGP++ validation and verification. According to [9], validation is the process to evaluate how accurate a model reflects a real-world phenomenon. In our case, instead of a real-world phenomenon we model BGP protocol. However, BGP models are not developed from scratch, rather Zebra BGP implementation is used. Thus, the validity of BGP++ is determined by the validity of Zebra BGP implementation and the validity of our integration methodology. Zebra is a routing software that has been used for some time by a sizable community. Also, our integration methodology replaces Zebra OS-related features
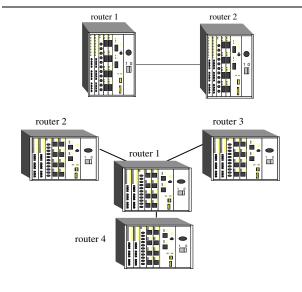
Figure 4. Testbed topologies.

	two routers	
	BGP++	testbed
KEEPALIVES Sent	1008	1006
KEEPALIVES Rcvd	1008	1006
UPDATES Sent	30	30
UPDATES Rcvd	30	30

Table 1. Numbers of exchanged messages in 2-router experiment

	four routers	
	BGP++	testbed
KEEPALIVES Sent	306	306
KEEPALIVES Rcvd	306	306
UPDATES Sent	180	180
UPDATES Rcvd	90	90

Table 2. Numbers of exchanged messages in 4-router expriment

with corresponding *ns-2* features. Since both *ns-2* and Zebra software have been widely used, we argue that BGP++ provides an accurate implementation of BGP protocol. Furthermore, verification of BGP++ is required. Verification is the process of evaluating how faithfully the implementation of a model matches the developer's intent [9]. For BGP++, this definition translates to how accurately our integration methodology was implemented. To verify BGP++ several scenarios, ranging from simple to more complicated, are conceived. The results are examined to determine if the protocols' expected behavior is observed. The tests are classified in four categories: basic behavior tests, policies related tests, logging facilities tests and advanced features tests. The following list enumerates the successfully tested features:

- Basic behavior tests: connection establishment, session termination, connection reset, route distribution, route selection algorithm.

- Policy related tests: route-maps, match set commands, ip access-lists, ip community-lists, ip as-path access-lists, ip prefix lists.

- Logging facilities tests: *show* command variants, binary dumps, debugging facilities.

- Advanced features tests: confederations, route-reflection, capability negotiation, soft reconfiguration, refresh capability.

Additional tests examine the behavior of BGP++ with reference to Zebra software. For this purpose, small testbeds of Zebra routers are set up. For each testbed a corresponding simulation of the same topology, and configuration is run. We demonstrate two experiments; the first consists of two peering routers, Figure, while the second consists of four peering routers connected in a star topology, Figure 4. For each experiment we take measurements in both real and simulation environment, over the same period, of the number of exchanged messages at router 1. The results on Tables 1 and 2 show that there is almost a perfect match between BGP++ behavior and Zebra behavior. The small difference in the keepalives count of the first experiment is due to BGP jittered timers and coarse-grained timing granularity of testbed experiments. The results illustrate that BGP++ effectively regenerates the behavior observed in the testbed environment.

3.3. Performance measurements

This subsection discusses BGP++ performance in terms of memory utilization. Extensive memory requirements is the primary limiting factor in large scale simulations. To measure BGP++ memory utilization a parametrized simulation scenario is created. The scenario is as follows: n routers are interconnected in a full mesh topology; each router is a separate AS that originates p prefixes. The routers establish sessions and exchange their configured prefixes. Thus, after steady-state is reached each router should have $p \times n$ prefixes in its routing table. Figure 5 illustrates how memory utilization increases with the size of the mesh and the number of configured prefixes. It should be noted that the main memory consumption comes from the represntation of the

routing tables. Increasing the number of perfixes, i.e. the size of the routing tables, limits the total size of the topology to few tenths of routers. In contrast, in the absence of large routing tables, simulations of few thousands of routers could be effectively performed.
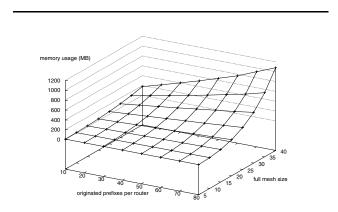

Figure 5. BGP++ memory utilization, when simulating a full mesh topology, versus mesh size and number of originated prefixes.

4. Parallel and Distributed BGP++

A major problem of simulation research is that conventional simulators exhaust the resources available in typical workstations resulting in simulations of average sized networks that lack the complexity of the simulated system. Results based on this type of simulations are usually not indicative of Internet wide trends. For this reason, there is a strong need for accurate large-scale simulations. Parallel and distributed simulation techniques have been exploited, namely PDNS [16] and SSFNet [3, 4] , to yield network simulators that achieve simulations of substantial size.

BGP++ provides support for parallel and distributed simulations through the functionality provided by the PDNS and the RTIKIT [5] toolkit. PDNS is an extension of *ns-2* simulator that allows a distributed simulation on a loosely coupled network of workstations. It makes minimal modifications in *ns-2* software. The RTIKIT provides support for global virtual time management, group data communications, and message buffer management. BGP++ uses PDNS and RTIKIT to distribute simulations on multiple machines. This feature allows to scale the size of the simulations, overwhelming the restrictions imposed by limited physical memory.

5. Conclusions

This work introduces a new simulation implementation of BGP for *ns-2*. BGP++ is a detailed implementation of the protocol that provides the user the capability to configure the simulated routers using the router configuration syntax used by Zebra *bgpd*. BGP++ design inherits credibility and reliability from the original software while requiring moderate development effort. The same approach can be used in similar projects to turn public domain software into simulation software.

In the future we consider extending the capabilities of BGP++ by supporting the latest version of Zebra software. Also, several memory reduction techniques will be considered to improve the scalability of the software. Seeing that the main memory consumption results from the simulation of routing tables, we intend to emphasize on techniques that reduce the size of the simulated routing tables. Finally, we intend to investigate BGP stability and scalability using BGP++, and examine how proposed BGP modifications affect interdomain routing.

References

[1] A. Broido and kc claffy. Analysis of RouteViews BGP data: policy atoms. In *Proceedings of NRDM workshop Santa Barbara*, May 2001.

[2] A. Broido and kc claffy. Complexity of global routing policies. In *Proceedings of IMA Special Workshop: Mathematical Opportunities in Large-Scale Network Dynamics*, Aug. 2001.

[3] J. Cowie, H. Liu, J. Liu, D. Nicol, and A. Ogielski. Towards realistic million-node internet simulations. In *International Conference on Parallel and Distributed Processing Techniques and Applications*, June 1999.

[4] J. H. Cowie, D. M. Nicol, and A. T. Ogielski. Modeling the global internet. *Computing in Science and Engineering*, Jan. 1999.

[5] R. M. Fujimoto, K. Permualla, and I. Tacic. Design of high performance RTI software. In *Distributed Simulation and Real-Time Applications 2000*, Aug. 2000.

[6] L. Gao and F. Wang. The extent of AS path inflation by routing policies. In *Proceeding of Global Internet 2002*, pages 188–195, Nov. 2002.

[7] R. Govindan and A. Reddy. An analysis of inter-domain topology and route stability. In *Proceedings of INFOCOM*, 1997.

[8] T. Griffin and B. Premore. An experimental analysis of BGP convergence time. In *Proceeding of the 9th International Conference on Network Protocols*, Nov. 2001.

[9] J. Heidemann, K. Mills, and S. Kumar. Expanding confidence in network simulation. Research Report 00-522, USC/Information Sciences Institute, April 2000. submitted for publication, IEEE Computer.

[10] K. Ishiguro. GNU Zebra. Software on-line: http://www.zebra.org.

[11] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed internet routing convergence. In *Proceedings of ACM SIG-COMM*, pages 175–187, 2000.

[12] C. Labovitz, A. Ahuja, R. Wattenhofer, and S. Venkatachary. The impact of internet policy and topology on delayed routing convergence. In *Proceedings of INFOCOM*, pages 537–546, 2001.

[13] S. McCanne and S. Floyd. The LBNL network simulator. Software on-line: http://www.isi.edu/nsnam, 1997. Lawrence Berkeley Laboratory.

[14] M. Pettersson. Performace Counters. Software on-line: http://sourceforge.net/projects/perfctr/.

[15] Y. Rekhter and T. Li. RFC 1771, Border Gateway Protocol 4, Mar. 1995.

[16] G. F. Riley, R. M. Fujimoto, and M. H. Ammar. Parallel/Distributed ns. Software on-line: www.cc.gatech.edu/ computing/ compass/ pdns/ index.html, 2000. Georgia Institute of Technology.

[17] G. Siganos and M. Faloutsos. BGP routing properties at large time scale. In *Global Internet Symposium*, Nov. 2002.

[18] H. Tangmunarunkit, R. Govindan, and S. Shenker. Internet path inflation due to policy routing. In *Proceeding of SPIE ITCom 2001, Denver 19-24*, pages 188–195, Aug. 2001.

[19] K. Varadhan, R. Govindan, and D. Estrin. Persistent route oscillations in inter-domain routing. *Computer Networks*, 32(1):1–16, 2000.