

Efficient Large-Scale BGP Simulations

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

Simulation has been the method of choice for analyzing large, complex, and highly volatile systems. One of these systems is the inter-domain routing infrastructure of the Internet. Despite the need for high quality Border Gateway Protocol (BGP) simulation tools, traditional BGP simulators have limitations either in their modeling fidelity or in their scalability. In this work we introduce BGP++, a scalable BGP simulator that employs state-of-art techniques to address the abstraction-scalability trade-off.

BGP++ builds on high quality software in network simulation, routing and parallel-distributed simulation to deliver a detailed yet scalable implementation of BGP. Moreover, with respect to the needs of researchers and operators, BGP++ has a CISCO-like configuration language, a seamless partitioning engine for parallel-distributed simulations and a configuration toolset that expedites common simulation configuration tasks.

Key words: BGP, network simulation

1 Introduction

Modeling and simulation analysis has played a key role in the field of computer networks. Typically, vendors and researchers evaluate prospective architectures and perform comprehensive “what if” analysis using simulation. Simulation is also used for parameter-tuning, problem diagnosis and performance optimizations. It is immensely important in research on large, complex and heterogeneous systems, like the Internet, where analytical models and laboratory testbeds do not capture the detail or the sheer volume of the system. One of these systems is the BGP infrastructure of the Internet.

BGP is the de-facto inter-domain routing protocol in the Internet. It is the “glue” that interconnects more than 16,000 Autonomous Systems (AS) of diverse geopolitical nature. In contrast to Interior Gateway Protocols (IGP), BGP is a policy-based

protocol. Business relationships and agreements between ASs determine how packets are routed.

The last few years the interdomain routing infrastructure has attracted substantial research interest. Some of the widely-researched problems are its slow convergence [1–5], policy conflicts [6,7], instability [8,9], misconfigurations [10], lack of security [11], table growth [12,13] and path inflation [14–16]. These problems necessitate fixing or even replacing the current routing architecture. Several examples of proposed new routing architectures can be found in the following references [17–23]. Yet, it is not clear how BGP should be modified.

The main reason the future of the routing architecture is dubious, is the lack of necessary tools to comprehensively understand the current infrastructure and to evaluate new alternatives. Measuring the performance of BGP is strongly prohibited by ownership and lack of measurement infrastructure. Analytical models of BGP are simplistic and do not capture the complexity, configurability and heterogeneity of the protocol. BGP simulators can only perform moderate-scale simulations of few hundreds of routers, from which we cannot draw reliable conclusions.

In this work we are concerned with building a simulation tool that will help researchers shed light on the design flaws of the current BGP infrastructure and evaluate the performance of new architectures. BGP++ is designed along the *scalability-realism* diptych, enabling for the first time large-scale as well as detailed BGP simulations. BGP++ is not a new bottom-up implementation of BGP, but it capitalizes on and benefits from existing high quality software on network simulation, routing and parallel-distributed simulation. The following software are the basic components on which we build BGP++:

- (1) ns-2 [24] is a discrete-event network simulator that serves as a common platform on which researchers can test and compare their proposals. ns-2 has evolved into the most widely-used simulator in networking literature. It includes numerous implementations of protocols with a special emphasis on TCP variants.
- (2) GNU Zebra [25] and its ancestor, Quagga [26]¹, is a detailed open-source implementation of BGP. It is used by a large community of providers, vendors and researchers for testing and experimenting with the protocol. It is also used for routing by small ASs that cannot afford buying expensive routers.
- (3) pdns [27] is the parallel-distributed version of the ns-2 simulator. It enables large-scale simulations by distributing the simulation model on multiple workstations, thereby granting more physical resources. It contrast to its counterpart, SSFnet [28], pdns supports both shared-memory multiprocessors and distributed-memory clusters of workstations. Support of distributed architectures offers more physical resources, overwhelming the limits inherent in par-

¹ BGP++ project predates Quagga project, for this reason we use Zebra.

allel architectures. Remarkably, pdns was recently used by Fujimoto et al. [29] to realize the largest network simulations ever, of more than 5 million network nodes.

Our contributions can be summarized as follows:

- (1) We develop and make publicly available [30] a packet-level BGP simulator on the widely-used ns-2 simulation platform.
- (2) We integrate the BGP implementation of Zebra into ns-2, making the minimum possible changes to the original software. We realize an accurate BGP simulator that supports most of the details of Zebra’s BGP implementation, including a CISCO-like configuration language.
- (3) We use and extend pdns to support parallel-distributed BGP simulations. We also evaluate the performance of alternative model partitioning algorithms for parallel-distributed simulations.
- (4) We identify the representation of BGP routing tables as the main source of memory consumption in BGP simulations. We introduce a compact routing table data structure that exploits the redundancy of information in BGP routing tables and realizes significant memory savings.
- (5) We propose and develop a simple generic technique to speed-up simulation trials using process checkpointing.
- (6) We survey recent advancements in measuring the Internet. We highlight measurement data and related models that should be explored to enhance the fidelity of BGP simulations.
- (7) We develop a seamless partitioning and configuration engine for parallel-distributed BGP simulations that hides the complexity of pdns configuration and brings parallel-distributed simulation closer to the general ns-2 user. We also develop an automatic generator of CISCO-like configuration. We combine these tools in a toolset that expedites common BGP simulation and configuration practices.

The remainder of this paper is organized as follows. Section 2 elaborates on previous efforts on BGP simulation. Section 3 describes in detail the development of BGP++. Section 4 introduces three techniques that we use to materialize large-scale BGP simulations. Section 5 surveys the developments in the field of Internet measurements with respect to BGP. Section 6 introduces a toolset to expedite configuration, partitioning and scheduling of BGP simulations. Finally, section 7 talks about future directions of our research.

2 Related Work

The last few years there has been a considerable work on BGP simulation research. The BGP model by Premore in the SSFnet [28] simulator was the first detailed

simulation model of BGP and is currently the most widely used BGP simulator. It supports most important BGP features and extensions. The main limitation of SSFnet is that it exhibits considerable memory demand, thereby preventing simulations larger than a few hundred of BGP routers.

Independently and in parallel with BGP++ a number of other efforts have developed BGP simulators. Among these efforts, the C-BGP [31] simulator and the work by Hao and Koppol [32] emphasize on large-scale BGP simulations. C-BGP is a BGP decision process simulator that, like BGP++, can read CISCO-like configuration files and can simulate large-scale topologies. Nevertheless, it only implements the BGP decision process, ignoring several details of the protocol, namely timers and BGP messages.

The recent work by Hao and Koppol [32] addresses the challenge of large-scale BGP simulations by ignoring the protocol stack below the application layer. Their simulator can perform large-scale experiments. Nevertheless, the relevant paper [32] does not discuss sufficiently the features of the simulator so as to develop a comprehensive picture of the work.

Other efforts parallel or posterior of BGP++ are [33–35] that ported the SSFnet BGP implementation in ns-2, JavaSim [36] and Genesis [37], respectively. In contrast to all previous works, BGP++ is the first simulator that makes large as well as detailed simulations feasible.

3 Modeling BGP

3.1 BGP++ development

The process of modeling a system is subject to the abstraction-scalability tradeoff. Higher abstraction results in more efficient models, which therefore are more scalable. On the other hand, detail is required to thoroughly capture the characteristics of the system. To create detailed simulation models we chose to incorporate the Zebra open-source BGP implementation into the ns-2 simulation environment. The main advantage of our approach is that the simulator inherits the detail, functionality and maturity of the original open-source software.

Zebra is written in C and implements three routing protocols: BGP, RIP and OSPF (see Figure 1). Each of the protocols has a separate daemon that can be run as a stand-alone process. An additional daemon, called the Zebra daemon, takes care of communication 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. We use the BGP daemon (bgpd)

and Zebra's library methods to build BGP++.

Throughout the integration process we take extra care to leave intact the fundamental logic of the code that implements BGP. However, Zebra and ns-2 are two intrinsically different software. The following list categorizes the high level differences we identify and exploit during the integration:

- (1) Use of C++ compared to C.
- (2) Use of discrete event scheduling algorithms compared to process-based scheduling algorithms.
- (3) Use of simulator's TCP implementation compared to BSD sockets.
- (4) Use of non-blocking routines compared to blocking routines.

Step 1: A fundamental difference between Zebra and ns-2 is that the former is designed to run one bgpd per process, while the latter needs to instantiate multiple BGP routers in the same process. The object-oriented paradigm of C++ enables ns-2 to instantiate multiple objects of a class. To meet this requirement we convert Zebra's C code into C++ and encapsulate Zebra's global variables into a C++ class. We create a BGP class in ns-2 that contains the original C code. The C functions are turned into C++ member functions and the global variables are turned into member variables.

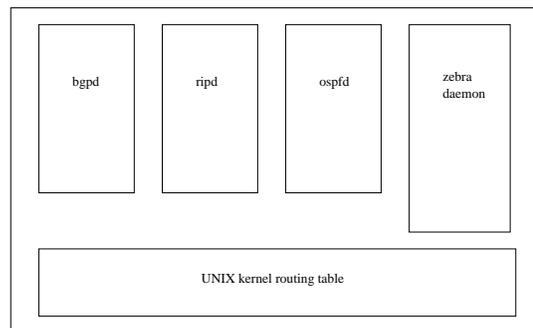


Fig. 1. GNU Zebra modular architecture

Step 2: The most challenging aspect of the integration is interleaving Zebra bgpd scheduler with ns-2 scheduler. 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 standard scheduling architecture. Scheduling varies with developer's design from simple to arbitrarily complex. Typical networking software has one or more blocking routines that blocks until an event triggers a response. To incorporate Zebra bgpd in the simulator, we modify Zebra scheduler to communicate with the queue scheduler. At a high-level the interleaved scheduling works as follows: whenever there is an event for Zebra bgpd, e.g., a start event, the simulator gives control to the bgpd to execute the associated code. The bgpd continues until the first blocking routine is reached; then, instead of blocking, it returns control to the scheduler. Note, that the bgpd should not block since the simulation does not run on wall-clock time. The bgpd is given control again when

the blocking routine would unblock. Events that could unblock the bgpd are read events, e.g., a packet arrival; write events, e.g., a buffer becomes writable; timer expiration events 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. When *select()* unblocks, execution continues by handling the event or events that caused the interrupt. *select()* is reached again through an infinite loop.

Our interleaved scheduling works as follows: at the start event ns-2 calls the simulated bgpd to make required initializations. The BGP finite state machine is entered and execution proceeds until the blocking routine *select()* is reached. Instead of blocking, the simulated bgpd enters an event for the calculated timeout value into the ns-2 queue scheduler and returns control to the simulator. If we disregard events that unblock the simulated bgpd, 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 ². For this reason, the simulator has to invoke the simulated bgpd upon a read or write event. In operating systems, read events occur when the TCP stream has new bytes available. In the simulator, upon a packet arrival, ns-2 cancels the future timeout event for the appropriate bgpd and gives control to it. Data packets are handed to the bgpd by the simulator's underlying TCP implementation. In Zebra, write events result from the fact that non-blocking output routines, namely *write()* and *writenv()*, do not copy the application buffer to the kernel output buffer immediately. 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 very small and we ignore it in our BGP simulations.

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 time, especially when their routing tables are large. For this reason, we implement a workload model that adds delay, representing the finite execution time of the CPU. We model the workload as follows: when a simulated bgpd 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 bgpd hangs for *busy-period* time before executing any following operation. During this period it does not respond to any events, e.g., packet arrivals. Instead, all events are buffered and executed in a FIFO order as soon as the bgpd resumes. Note that each of the buffered events will result in new operations and *busy-period* intervals. We implement two

² User interrupts are treated as read events, since user communication is done through a telnet interface.

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 the CPU time that was allocated for the operation that just completed by the workstation the simulation is running on. Using a kernel patch [38], BGP++ monitors the number of cycles its operations consume. Then, it calculates the *busy-period* as the product of the CPU clock frequency and the count of consumed cycles.

Step 3: The third step is to substitute the BSD socket API with the corresponding TCP implementation of ns-2. We choose to use the *ns-2 FullTcp* implementation and modify *FullTcp* to simulate the socket API. Our modifications notify the application that started the *FullTcp* instance as soon as the connection moves from *SYN_RCVD* or *SYN_SENT* to *ESTABLISHED* and from *ESTABLISHED* to *CLOSE_WAIT*. The first two transitions correspond to the BSD sockets non-blocking *connect()* and *accept()*, respectively. In both cases they notify the application that the three way hand-shake has completed successfully. The third transition notifies the application upon passive connection termination.

Step 4: The last step to introduce Zebra routing software in ns-2 simulation environment is to replace system calls with corresponding simulator functions, replace wall-clock time functions with simulation time functions and remove unnecessary code. Zebra supports a telnet interface that is used to configure or query the routing daemons at run-time. We remove the telnet interface since it is not useful in a simulation environment. Moreover, we replace the functionality of the telnet interface with a new interface that allows to query and reconfigure the simulated routers at run-time.

3.2 BGP++ validation and verification

According to [39], 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. However, we do not develop our BGP simulator from scratch, but we use pre-existing software. Thus, the validity of BGP++ is determined by the validity of Zebra's BGP implementation, the validity of ns-2 simulator and the validity of our integration methodology. Both Zebra and ns-2 open-source software have been used for a significant time by large communities. Also, our integration methodology replaces Zebra OS-related functions with corresponding ns-2 functions. Since both ns-2 and Zebra software have been widely used, we argue that BGP++ provides an accurate implementation of BGP.

Furthermore, verification of BGP++ is required. Verification is the process of evaluating how faithfully the implementation of a model matches the developer's intent [39]. For BGP++, this definition translates to how accurately we implement our

integration methodology. To verify BGP++, we develop several scenarios, ranging from simple to more complicated, and perform simulations testing the behavior of BGP++. For each scenario we examine the results and make sure that the observed behavior is in line with the expected behavior. We develop test scenarios of the following types: basic behavior tests, policy related tests, logging facilities tests and advanced features tests. The following list enumerates the tested features:

- Basic behavior tests: connection establishment, session termination, connection reset, route distribution, route selection algorithm.
- Policy related tests: *route-maps*, *match* and *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.

We also perform additional tests to compare the behavior of BGP++ with the original unmodified Zebra software. For this purpose, we setup small testbeds of Zebra routers and compare the observed behavior with corresponding simulations of the same topology and configuration. The results demonstrate that BGP++ effectively and accurately models the behavior observed in the testbed environment. For a more detailed description of these tests refer to [40].

3.3 BGP++ configuration

The development approach adapted for BGP++ preserves most of the features of the original software, among which the configuration language. Each simulated BGP router parses a configuration file, written in the configuration language used by Zebra routers. The Zebra configuration language is very similar to the well-known configuration language used by CISCO routers. For example, configuration structures like *route-maps*, *access-lists* and *prefix-lists* are parsed by the simulated routers. Moreover, the functionality available in the Zebra software through the telnet interface is maintained in BGP++ through a new TCL interface. The user can instruct a simulated router at any point during the simulation to execute a given command, like *show ip bgp*, which would otherwise be entered through the telnet interface. A more detailed description of BGP++ configuration can be found in [40].

4 Scaling BGP++

The need for large-scale BGP simulations stems from the scale of the BGP infrastructure. More than 16,000 ASs use BGP, including a number of global providers that administrate thousands of internal routers. Also, the advent of the Internet measurements research the last few years has enhanced the information known about BGP, making it possible to construct more detailed and realistic simulations. However, this comes at the cost of more simulation resources, pronouncing the need for scalable BGP simulators. Efficient simulations are also vital, especially in applications on network control and debugging, but also when real-time guarantees are required. In this section we describe how we make *large-scale* BGP simulations feasible.

We first show that the memory demand for BGP simulations is driven by the memory required to represent routing tables. To address this problem we introduce a routing table representation data structure, which exploits the redundancy of routing table information across BGP instances. Our experiments indicate that our compact routing table data structure results in up to 62% memory reduction in the total memory required for the simulation. Using the compact routing table data structure we make proof-of-concept simulations of up to 4,000 simulated bgpd in a single workstation with 2GB of memory.

Next, we integrate BGP++ with pdns to make parallel-distributed BGP simulations possible and evaluate the performance of different model partitioning algorithms for parallel-distributed BGP simulations.

In Section 4.3 we introduce a simple and efficient technique to speed up execution time in simulation trials using process checkpointing. Our technique is generic, however we implement it on BGP++ and evaluate it using BGP simulations.

4.1 Efficient Routing Table Representation

The memory consumption of large-scale BGP simulations depends mainly on the following parameters: the size of the topology, the size of the routing tables, the number of neighbors per router, the simulation dynamics and the simulator memory footprint. A BGP router maintains three Routing Information Bases (RIB): the Adj-RIB-in, the Loc-RIB and the Adj-RIB-out [41]. The Adj-RIB-in stores routes received by neighboring BGP routers, the Loc-RIB stores routes selected by the decision process and the Adj-RIB-out stores routes advertised to other BGP routers. Let N denote the number of BGP routers in a simulation; p the average number of prefixes originated per router; r the average number of neighbors per router; α , β , γ and δ proportionality constants. Then, the following formula approximates the total memory demand for a BGP simulation:

$$Memory \approx \alpha N + \beta p N^2 + \gamma p N^2 r + \delta \quad (1)$$

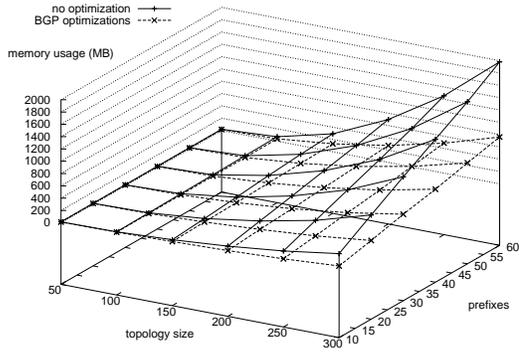
Term αN represents the memory cost to initialize the BGP routers or, in other words, the memory cost for routers with empty routing tables; term $\beta p N^2$ accounts for the Loc-RIBs memory consumption, assuming that each router after convergence has pN entries in the routing table; term $\gamma p N^2 r$ is the worse possible memory consumption for the Adj-RIB-ins and Adj-RIB-outs, assuming that each router receives pN routes from each of its r neighbors ³; term δ represents the simulator footprint. It follows that for $r = N$ the memory demand is $O(N^3)$ and is driven by the routing table term $\gamma p N^2 r$.

Our compact routing table data structure relies on the observation of redundant information in BGP routing tables. We find that different routing entries in BGP tables often share common information for one or more fields. The redundancy lies across three dimensions: 1) within the same routing table, across different entries 2) within the same router, across different routing tables (Adj-RIB-in, Loc-RIB, Adj-RIB-out) and 3) within the same simulation, across different routers. For each routing table entry a BGP router stores several BGP attributes. For example, a routing table entry in BGP++ stores the following attributes: AS path, origin, next hop, local preference, multi-exit discriminator (MED), communities, extended communities and *unknown* ⁴. The attributes account for most of the memory required for a table entry. For this reason, we create a data structure that shares common attributes among different routing table entries.

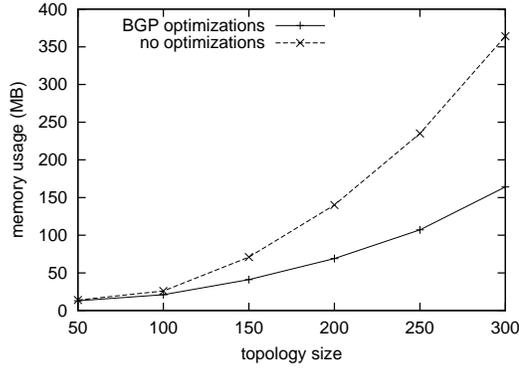
We associate each route with a structure *struct attr* that has one entry for each of the possible attributes. For attributes with size less or equal to 32 bits the entry is the value of the attribute, otherwise a pointer to another data structure specific to the attribute in question. For example, *struct attr* contains the value of the MED attribute, which is a 32-bit unsigned integer. In contrast, it contains a pointer to an AS path data structure, which can be of arbitrary size. The AS path data structure is associated with a reference counter and stored in a global hash table. There is one global hash table for each type of attribute with size greater than 32 bits. A router, after allocating and initializing memory for a new attribute, it searches the relevant hash table for the newly allocated attribute. In case of a match, the allocated memory is deallocated and a reference to the attribute in the hash table is used. In addition, it increments the reference counter, which is used to track the number of pointers to the attribute. If the attribute is not found in the hash table, then the router creates an entry for the attribute, so that subsequent searches will succeed. If a router wishes to remove an attribute, it decrements its reference counter. When the reference counter becomes zero the attribute is removed from the hash table

³ BGP policies tends to decrease memory demand by limiting the number of paths known to a router.

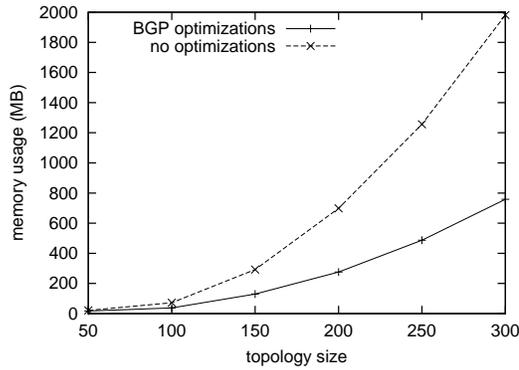
⁴ BGP specifications require to transit unknown attributes.



(a) Memory usage versus topology size (N) and originated prefixes per router (p)



(b) Cross-section of 2(a) for 10 originated prefixes per router



(c) Cross-section of 2(a) for 60 originated prefixes per router

Fig. 2. Memory savings by sharing information among information bases

and deallocated. This structure handles redundancy by creating a centralized pool of attributes, which are shared among different routers, tables and table entries. Similarly, a second level hash table is used to store and share *struct attr* structures.

We evaluate the effectiveness of our compact routing table data structure using simulation. For the simulation topology we use a random connected subgraph of the Internet AS topology as seen from RouteViews [42] of size N ; each AS has a single BGP router that originates p prefixes. The policies between the ASs are inferred using the heuristics described in [43]. The latter classifies policies in three categories: provider-to-customer, peer-to-peer and sibling-to-sibling. For simplicity we modify sibling-to-sibling relationships to peer-to-peer⁵. The inferred relationships are configured using BGP communities as follows: a provider advertises to a customer all the routes it knows; a customer advertises to a provider routes that are either locally originated or learned by its customers; likewise a peer advertises to a peer routes that are either locally originated or learned by its customers. These configurations are consistent with typical operator practices [43]. We set the delay and the data rate of all links to 10ms and 10Mbps, respectively and we ignore the processing workload of the routers. Finally, we run each simulation until the system reaches steady state, i.e., no updates are exchanged. For the remainder of this paper we refer to this setup as $Internet(N, p)$.

Figure 2(a) illustrates the total memory consumption with and without routing table optimizations versus N and p in an $Internet(N, p)$ setup. Cross-sections of the figure are shown in Figures 2(b) and 2(c) for 10 and 60 originated prefixes per router, respectively. The highest memory reduction achieved is 62% with a mean of 47% with respect to the total memory. Note that these numbers are conservative since the total memory consumption is the result of several sources of memory demand besides the routing tables. The memory reduction with respect to the memory required for the unmodified routing tables is even larger.

We also adapt an existing scheme, called Nix-Vector routing [44], to further reduce the routing tables memory consumption. Nix-Vector routing reduces the required memory to represent the Forwarding Information Bases (FIB) by computing routes on demand, as needed, rather than pre-computing all possible routes. We evaluate the memory we save by using Nix-Vector routing as compared to ns-2 default static routing in an $Internet(N, 1)$ setup. Figure 3 illustrates that Nix-Vector routing can further reduce the total memory consumption. The peak memory reduction is 36% and the mean 22%. For p larger than 1, the memory required for the BGP routing tables dominates, overshadowing the memory savings of Nix-Vector routing.

To evaluate the scalability of BGP++ we conduct proof-of-concept large-scale simulations. We effectively simulate up to 4,000 Zebra bgpd in a single workstation with 2GB of physical memory. The simulation setup is an $Internet(4000, 1)$. The

⁵ This modification has no impact on evaluating the discussed techniques.

size of our simulations are an order of magnitude larger than the corresponding simulations we are able to perform using SSFnet under the same configuration on the same workstation.

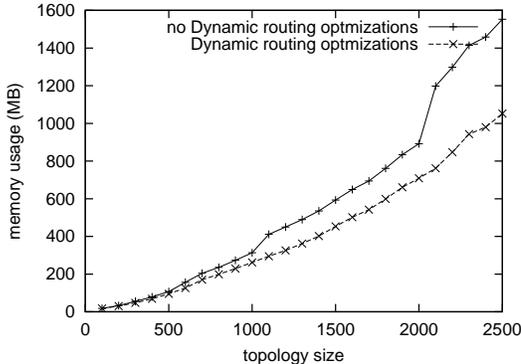


Fig. 3. Memory savings by using Nix Vector routing

4.2 Parallel Distributed BGP Simulations

The use of parallel-distributed techniques has been exploited to lead to faster and larger network simulations in tools like pdns and SSFnet. The pdns simulator is a space-parallel, conservative synchronization simulator derived from ns-2. As a space-parallel simulator, the simulated topology is partitioned and distributed on different processes (federates on pdns terminology) on different computing platforms. Also, the conservative synchronization protocol bounds the lag of the simulation time between federates to insure causal consistency. The RTIKIT [45] library provides support for global virtual time management, data communications and message buffer management between multiple federates. We use the RTIKIT library to extend pdns. Our modifications enable to create BGP sessions between simulated bgpd in different federates.

A critical decision when constructing a distributed network simulation is the distribution of the topology model. Proper distribution has significant impact on the execution time of the simulation. In pdns, each simulator instance is assigned a different part of the simulated topology. Consequently, distribution of the model becomes a graph partitioning problem. The simulation is divided into epochs, which are intersected by synchronization intervals. During the synchronization interval, cross-simulator events are delivered. Let ET_{ij} and CT_{ij} denote the execution and communication time of federate i at epoch j , respectively. Also, let ST_j be the synchronization time at epoch j . Then, the execution time of the event processing phase of a simulation is:

$$ET = \sum_0^{N-1} (\max_i \{ET_{ij} + CT_{ij}\} + ST_j) \tag{2}$$

where N is the number of epochs ⁶. Partitioning has a direct impact on the terms ET_{ij} , CT_{ij} , ST_j and N of equation 2. In particular, partitioning should:

- Balance the processing workload to minimize the maximum ET_{ij} over i .
- Minimize the maximum communication load between any two federates. This has a direct impact on both CT_{ij} and ST_j since cross-simulator events are delivered during synchronization.
- Maximize the length of the epochs to increase parallelism and reduce N .

A challenging objective in topology decomposition is to accurately load balance a distributed simulation. This is because we cannot know in advance the processing load of a given partition. Recent works by Yocum et al. [47] and Liu et al. [48] propose methods to predict the load of a partition using static configuration information. Their work concentrates on distributed emulation, but their techniques are also applicable in the context of space-parallel distributed simulations. Both schemes model the processing and communication load as weights of vertices and edges of a graph and use graph partitioning algorithms to find good partitions. In particular, the graph partition problem is formulated as follows:

Given an undirected graph $G = (V, E)$, where V the set of vertices and E the set of edges find a partition of G in k parts that minimizes the edge-cut under the constrain that the sum the vertices' weights in each part is balanced.

The processing workload associated with a simulated BGP router consists of maintenance workload, which refers to maintenance tasks such as sending keep-alive messages, and dynamic workload, which refers to dynamic tasks such as update message processing. We evaluate two approaches to assign weights to BGP routers. In the *degree-weight* approach, we assume that the CPU cycles spent for a simulated router are proportional to the number of neighbors of the router. In this case we ignore BGP dynamics and asymmetries that arise from policies and configuration and capture the maintenance workload. Consequently, the weight of each vertex is set to the degree of the vertex. In the *equal-weight* approach, all routers are equally weighted, hence the algorithm partitions the topology into parts that have roughly the same number of BGP routers. In both approaches, we treat links that cross federates equally in terms of communication load. Therefore we do not apply distinct weights. This approximation captures faithfully the housekeeping traffic, but it does not capture traffic asymmetries due to BGP dynamics and policies.

The above graph partitioning problem is NP-complete. However, many algorithms have been developed to find reasonably good partitions [49]. We first evaluate the performance of different partitioning algorithms implemented in the Chaco [50] and Metis [51] graph partitioning packages. We compare the following algorithms: Chaco multilevel Kernighan-Lin (KL), Chaco spectral, Chaco linear, METIS multilevel KL and Chaco random. In the latter, vertices are assigned randomly in parti-

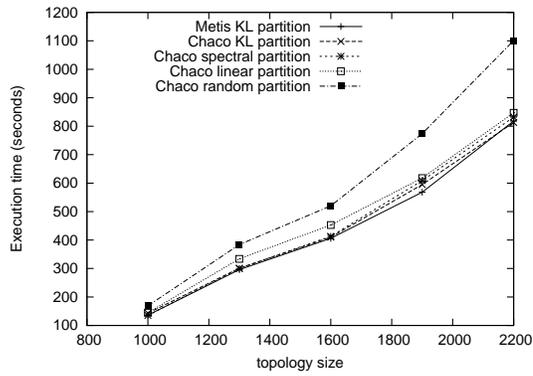
⁶ A similar formalization can be found at [46].

tions subject to the balance constraint, thus we use it as a worse case reference. To evaluate the partitioning algorithms we use a variant of the *Internet*($N, 1$) setup denoted as *Internet*($N, 1, 200$), in which the simulation time is fixed to 200 seconds. We assign weights with the degree-weight and equal-weight approaches and partition the graph into $k = 2$ federates. Figure 4(a) shows the execution time for the equal-weight case. We observe that the examined algorithms yield similar performance, while the savings with respect to the Chaco random algorithm increase with the model size. We find that METIS multilevel KL algorithm renders slightly better performance for both degree-weight and equal-weight partitioning. For this reason, we select it for the rest of our experiments.

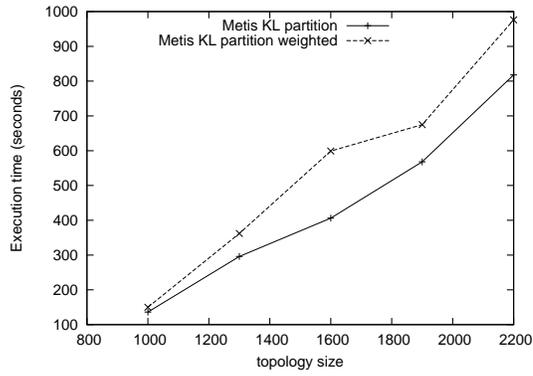
Next, we compare the degree-weight to the equal-weight weighting approach. Figure 4(b) illustrates that the degree-weight approach results in worse performance than the equal-weight approach in an *Internet*($N, 1, 200$) setup. The same is also illustrated in figure Figure 4(c) for an *Internet*($N, 0, 200$) setup. We discover that the worse performance results from the fact that the partitioning algorithm finds worse solutions in terms of edge-cut when the graph is weighted. Table 1 shows the edge-cut of the partitions found by the METIS multilevel KL algorithm for the degree-weight and equal-weight approaches, with $k = 2$. The increase of the edge-cut for the degree-weight approach results in a significant communication volume that lengthens the event processing as well as the synchronization phase of the simulation. This corresponds to the CT_{ij} and ST_j terms of equation 2. We verify that similar results hold for larger number of partitions. We speculate that the moderate performance of the partitioning algorithm for degree-weighted graphs is because of the power-law properties of the AS graph. The power-law degree distribution of the Internet AS topology, results in power-law distributed vertex weights, which make compliance to the balance constrain complex and produce worse edge-cuts.

Topology size	partition edge-cut	
	equal-weight	degree-weight
1000	33	119
1300	62	451
1600	90	442
1900	113	484
2200	175	719
2500	218	832

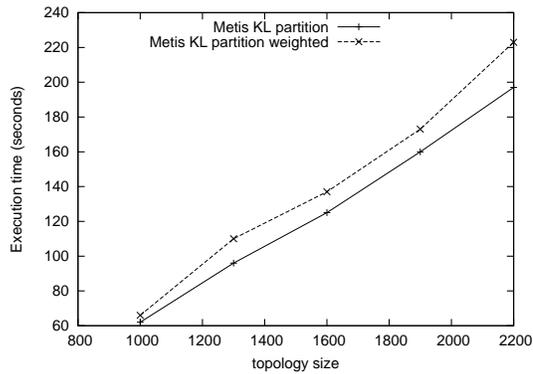
Table 1
Edge-cut found by METIS multilevel KL algorithm for degree-weighted and equal-weighted graphs, split into two parts



(a) Total execution time using different partitioning algorithms



(b) Total execution time of degree-weight versus equal-weight partitioning in an *Internet(N, 1, 200)* setup



(c) Total execution time of degree-weight versus equal-weight partitioning in an *Internet(N, 0, 200)* setup

Fig. 4. Partitioning of parallel-distributed simulations

4.3 Execution Time Optimizations

In this section we describe a simple and efficient technique to reduce simulation time in the presence of repeated simulation trials. Simulation trials are required to make simulation based results statistically significant. For example assume a simulation in which we want to measure BGP convergence time after a BGP withdrawal. To make a reliable claim, it is necessary to repeat the same simulation many times and calculate the average of the convergence time.

Our execution time optimization exploits determinism in simulations to avoid repeating the same trials. Any simulation starts with a deterministic period. The length of this period depends on the particular configuration, but at the minimum it lasts until the simulation is randomized, e.g., the random number generator is used for the first time. We exploit the determinism of a simulation by saving an image of the simulation process just before the simulation is randomized. The process of saving an image of a process is called checkpointing a process. Then, we start each trial from the saved image, circumventing the deterministic period. If r is the ratio of the deterministic period length to the total execution time and t the number of trials, the speedup is:

$$Speedup = \frac{ET_{normal}}{ET_{optimized}} = \frac{1}{1 - r + \frac{r}{t}} \quad ^7$$

For $r = 0.2$ and $t = 50$ the speedup is 1.244.

We extend BGP++ to support process checkpointing using Condor [52], a workload management system that supports checkpointing and restarting a process. We implement a command to request a checkpoint of the simulator process as soon as the initialization has completed. The initialization phase includes construction of network objects, calculation of static routes (if needed), initialization of BGP data structures and parsing of BGP configuration files. We also implement a second command that enables the user to checkpoint the simulator process at any point during the simulation. In this case, the user has the option to change the configuration of the simulation just after the simulation process restarts. This way, multiple scenarios can be forked from the same image.

We examine the execution time savings of this approach in an *Internet*($N, 1$) setup. Figure 5 depicts the total execution time and the initialization time for different N . The mean r is 0.17. The total execution time depends on the dynamics of the BGP system, which in turn depend on the seed of the simulation. This variation explains why the total execution time curve is not smooth. In a second run, where the second optimization of section 4.1 is deployed, r drops to 0.08 since the calculation of static routes during initialization is bypassed.

⁷ It is assumed that the overhead to restart a simulation from a saved image is negligible.

Using process checkpointing more aggressively we can get even greater execution time savings. For example, one could save the state of a BGP simulation once it has converged and then run multiple experiments to measure the effect of a BGP withdrawal.

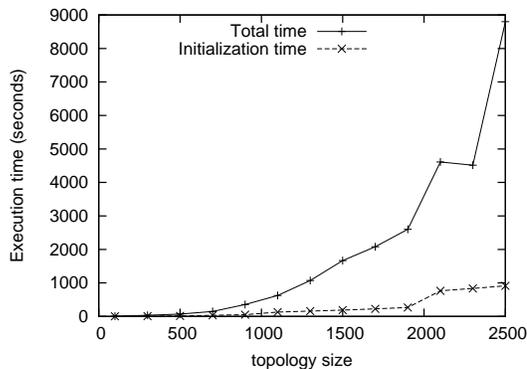


Fig. 5. Total execution time and initialization time versus topology size.

5 How to simulate BGP

Development of realistic simulation models is a challenging endeavor. The reliability of conclusions drawn from simulation analysis depends highly on the accuracy and the correctness of the simulation models. The accuracy of these models can be enhanced by exploring Internet measurements. In this section we survey the data and models available from Internet measurement studies that can be used to construct realistic BGP simulations.

A BGP simulation may include one or more of the following:

- Internet topologies at the AS and router level.
- Routing policies between ASs.
- Models of iBGP connectivity patterns. Connectivity patterns include the topological properties of the overlay iBGP network as well as iBGP design alternatives, e.g., route reflection, BGP confederations, full mesh.
- Router processing delay models.
- Link failure models.

Other information such as IGP protocols used, IP blocks allocated, links' delay and capacity, BGP traffic patterns may also be necessary depending on the goals of the specific simulation analysis.

In the recent years there has been a considerable effort in expanding our understanding of BGP by deploying measurement equipment in the Internet. The RouteViews project provides a union of BGP tables from a number of Internet Service Providers (ISP), including some of the most well-known tier 1 ISPs. The BGP

tables provide a fairly accurate map of the Internet AS topology. However, BGP policies hide many links from the tables [53,54]. Similar BGP table dumps can be found in RIPE [55] or in the route servers listed in [56]. Other sources of AS topologies are BGP updates [53,57] and Whois databases. AS topologies are also available by CAIDA that maps skitter [58] traceroute IP addresses to AS numbers [59]. Router level topologies are also available from many projects [58,60,61], although they are of questionable completeness [62]. As a consequence the work on modeling router level topologies is in preliminary stages [63]. Business relationships between ASs and BGP policies can be inferred from BGP data with the following heuristics [64,43,65–67]. Development of validation techniques and inference of more complex policies, e.g., partial transit [68], are still to be explored. To our knowledge, there are no models of iBGP connectivity patterns or monitoring projects that can provide this information. BGP processing delay has been shown through simulation studies to bear impact on BGP convergence time [69]. Initial steps to characterize and model BGP processing time include [70] and [71]. Preliminary models of link failures, based on measurements from an Internet backbone network, were introduced in [72].

6 BGP++ configuration toolset

We develop a toolset to expedite typical simulation configuration tasks. The simulation toolset can read simple user input files, generate CISCO-like configuration files, partition parallel-distributed simulations, generate configuration for pdns and schedule multiple simulation runs. Figure 6 illustrates a block diagram of the software architecture of the toolset. There are three main components: a configuration generator, a partitioner and a scheduler.

The configuration component takes as input an AS topology that has been annotated with AS relationships. The relationships between ASs in the Internet can be derived using one of the heuristics discussed above. Then, it creates CISCO-like configuration files for BGP++. BGP communities and filters are used to translate business relationships to appropriate BGP policies. The user has the option to specify many different parameters like the topology size or the number of announced routes.

The partitioner handles partitioning for parallel-distributed simulations. The user can choose among different partitioning algorithms supported through METIS and Chaco graph partitioning packages. Given a sequential configuration file for ns-2 it generates configuration for pdns and its BGP extensions. The partitioner provides a seamless interface to the parallel-distributed simulator, hiding from the user the complicated configuration of pdns.

Finally, the scheduler handles scheduling of multiple simulation runs. We imple-

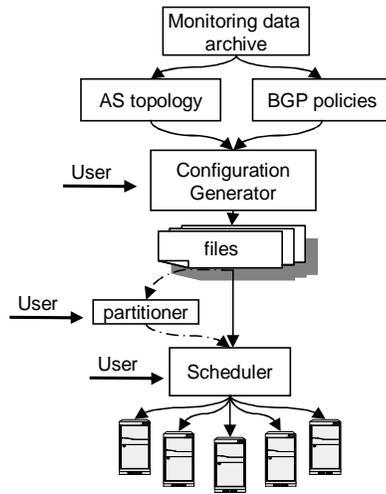


Fig. 6. Block diagram of configuration toolset

ment a master slave architecture, in which a master process schedules and distributes simulations to slave machines, taking into account available memory, existing CPU workload and other user specified criteria.

7 Conclusions and Future Work

In this work we develop BGP++: a *scalable* and *detailed* BGP simulator that is built on top of high quality software in network simulation (ns-2), routing (Zebra) and parallel-distributed simulation (pdns).

BGP research has ample fertile ground for simulation-based analysis. BGP++ opens new directions in using simulation to improve our understanding of the Internet interdomain infrastructure.

Evaluating the future performance of the interdomain infrastructure is a problem in which simulation is essential. With the current growth rates, Internet will have more than 34,000 ASs by 2010. This figure may be even larger given the accelerating spread of the Internet outside G7 countries. It is unknown how BGP performance and problems will be shaped by the increased complexity and scale. Large-scale BGP simulations are necessary to shed light on the scalability limits of BGP.

Finally, a systematic framework for BGP network control, configuration testing and problem diagnosis is another application of network simulation. We envision a simulation interface that replicates a real network, providing practical tools to check topological or configuration changes and to diagnose unforeseen problems.

Acknowledgments

This work is supported in part by NSF under contract numbers ANI-9977544 and ANI-0136969; and in part by DARPA under contract number N66002-00-1-8934.

References

- [1] C. Labovitz, A. Ahuja, A. Bose, F. Jahanian, Delayed Internet routing convergence, in: Proceedings of ACM SIGCOMM, 2000, pp. 175–187.
- [2] C. Labovitz, A. Ahuja, R. Wattenhofer, S. Venkatachary, The impact of Internet policy and topology on delayed routing convergence, in: Proceedings of INFOCOM, 2001, pp. 537–546.
- [3] D. Pei, X. Zhao, L. Wang, D. Massey, A. Mankin, S. F. Wu, L. Zhang, Improving BGP convergence through consistency assertions, in: Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Society (INFOCOM-02), Vol. 2, IEEE Computer Society, Piscataway, NJ, 2002, pp. 902–911.
URL citeseer.nj.nec.com/pei02improving.html
- [4] A. Bremler-Barr, Y. Afek, S. Schwarz, Improved BGP convergence via Ghost Flushing (2003).
URL citeseer.nj.nec.com/571575.html
- [5] D. Pei, M. Azuma, N. Nguyen, J. Chen, D. Massey, L. Zhang, BGP-RCN: Improving BGP Convergence Through Root Cause Notification, Tech. Rep. TR-030047, UCLA Department of Computer Science (Oct 2003).
- [6] T. Griffin, G. Willfong, A safe path vector protocol, in: Proceedings of IEEE INFOCOMM, 2000.
- [7] T. G. Griffin, A. D. Jaggard, V. Ramachandran, Design principles of policy languages for path vector protocols, in: Proceedings of ACM SIGCOMM, 2003.
- [8] K. Varadhan, R. Govindan, D. Estrin, Persistent route oscillations in inter-domain routing, *Computer Networks* 32 (1) (2000) 1–16.
- [9] R. Govindan, A. Reddy, An analysis of inter-domain topology and route stability, in: Proceedings of INFOCOM, 1997.
- [10] R. Mahajan, D. Wetherall, T. Anderson, Understanding BGP misconfiguration, in: ACM SIGCOMM, 2002.
- [11] S. Kent, C. Lynn, K. Seo, Secure Border Gateway Protocol (Secure-BGP), *IEEE Journal on Selected Areas in Communications* 18 (4) (2000) 582–592.
- [12] G. Huston, BGP table statistics.
- [13] T. Bates, Routing table history, <http://www.employees.org:80/tbates/cidr.plot.html>.

- [14] N. Spring, R. Mahajan, T. Anderson, Quantifying the causes of path inflation, in: ACM SIGCOMM, 2003.
- [15] L. Gao, F. Wang, The extent of AS path inflation by routing policies, in: Proceeding of Global Internet 2002, 2002, pp. 188–195.
- [16] H. Tangmunarunkit, R. Govindan, S. Shenker, Internet path inflation due to policy routing, in: Proceeding of SPIE ITCOM 2001, Denver 19-24, 2001, pp. 188–195.
URL <http://www.isi.edu/~hongsuda/publication/ITCOM2001.ps>
- [17] I. Castineyra, N. Chiappa, M. Steenstrup, The Nimrod Routing Architecture, IETF, RFC 1992 (1996).
- [18] F. Kastenholz, ISLAY: A New Routing and Addressing Architecture, IRTF, Internet Draft (2002).
- [19] X. Yang, NIRA: A new Internet routing architecture, ph.D. Thesis, Massachusetts Institute of Technology (September 2004).
- [20] P. Verkaik, A. Broido, kc claffy, R. Gao, Y. Hyun, R. van der Pol, Beyond CIDR aggregation, cAIDA Technical Report TR-2004-1 (February 2004).
- [21] L. Subramanian, M. Caesar, C. T. Ee, M. Handley, M. Mao, S. Shenker, I. Stoica, Towards a next generation inter-domain routing protocol, in: Proceedings of ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets-III), 2004.
- [22] S. Agarwal, C.-N. Chuah, R. Katz, OPCA: Robust interdomain policy routing and traffic control, in: IEEE OPENARCH, 2002.
- [23] A. Snoeren, B. Raghavan, Decoupling policy from mechanism in internet routing, in: Proceedings of ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets-II), 2003.
- [24] S. McCanne, S. Floyd, The LBNL network simulator, <http://www.isi.edu/nsnam>, Lawrence Berkeley Laboratory (1997).
- [25] K. Ishiguro, GNU Zebra, <http://www.zebra.org>.
- [26] P. Jakma, V. Jardin, A. Schorr, H. Tepper, G. Troxel, Quagga Routing Suite, <http://www.quagga.net>.
- [27] G. F. Riley, R. M. Fujimoto, M. H. Ammar, Parallel/Distributed ns, www.cc.gatech.edu/computing/compass/pdns/index.html, Georgia Institute of Technology (2000).
- [28] J. H. Cowie, D. M. Nicol, A. T. Ogielski, Modeling the global Internet, Computing in Science and Engineering.
- [29] R. M. Fujimoto, K. S. Perumalla, A. Park, M. A. H. Wu, G. F. Riley, Large-Scale Network Simulation. How Big? How Fast?, in: Proceedings of Eleventh International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'03), 2003.

- [30] X. Dimitropoulos, G. Riley, BGP++ webpage, <http://www.ece.gatech.edu/research/labs/MANIACS/BGP++/>.
- [31] S. Tandel, C. de Launois, C-BGP Home Page, <http://cbgp.info.ucl.ac.be/>.
- [32] F. Hao, P. Koppol, An Internet scale simulation setup for BGP, *Computer Communication Review* 33 (3) (2003) 43–57.
- [33] T. D. Feng, R. Ballantyne, L. Trajkovi, Implementation of BGP in a network simulator, in: *Proc. of Advanced Simulation Technologies Conference 2004 (ASTC'04)*, 2004.
- [34] Towards a BGP model in JavaSim, <http://www.info.ucl.ac.be/bqu/jsim/>.
- [35] B. K. Szymanski, Y. Liu, R. Gupta, Parallel network simulation under distributed Genesis, in: *In Proceedings of ACM/IEEE/SCS of Workshop on Parallel and Distributed Simulation (PADS)*, 2003.
- [36] J.-Y. Tyan, C.-J. Hou, JavaSim: A component-based compositional network simulation environment, in: *Proceedings of the Western Simulation Multiconference, Communication Networks And Distributed Systems Modeling And Simulation*, 2001.
- [37] Genesis Home Page, <http://www.genesis-sim.org/GENESIS/>.
- [38] M. Pettersson, Performace Counters, <http://sourceforge.net/projects/perfctr/>.
- [39] J. Heidemann, K. Mills, S. Kumar, Expanding confidence in network simulation, Research Report 00-522, USC/Information Sciences Institute (April 2000).
URL <http://www.isi.edu/~johnh/PAPERS/Heidemann00c.html>
- [40] X. Dimitropoulos, G. Riley, Creating realistic BGP models, in: *Proceedings of Eleventh International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'03)*, 2003.
- [41] Y. Rekhter, T. Li., RFC 1771, Border Gateway Protocol 4 (Mar. 1995).
- [42] University of Oregon Route Views Project.
URL <http://www.routeviews.org/>
- [43] L. Gao, On inferring Autonomous System relationships in the Internet, in: *Proc. IEEE Global Internet Symposium*, 2000.
URL citeseer.nj.nec.com/gao00inferring.html
- [44] G. F. Riley, M. H. Ammar, E. W. Zegura, Efficient routing using nix-vectors, in: *2001 IEEE Workshop on High Performance Switching and Routing*, 2001.
- [45] R. M. Fujimoto, K. Permualla, I. Tacic, Design of high performance RTI software, in: *Distributed Simulation and Real-Time Applications*, 2000.
- [46] D. M. Nicol, Scalability, locality, partitioning and synchronization in PDES, in: *Proceedings of the Parallel and Distributed Simulation Conference (PADS'98)*, 1998, banff, Canada.

- [47] K. Yocum, E. Eade, J. Degeys, D. Becker, J. Chase, A. Vahdat, Toward scaling network emulation using topology partitioning, in: Proceedings of Eleventh International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'03), 2003.
- [48] X. Liu, A. A. Chien, Traffic-based load balance for scalable network emulation, in: in Proceedings of the ACM Conference on High Performance Computing and Networking, 2003.
- [49] R. Ponnusamy, N. Mansour, A. Choudhary, G. C. Fox, Graph contraction and physical optimization methods: a quality-cost tradeoff for mapping data on parallel computers, in: In International Conference of Supercomputing, 1993.
- [50] B. Hendrickson, R. Leland, The Chaco user's guide (1994).
URL citeseer.ist.psu.edu/hendrickson94chaco.html
- [51] G. Karypis, V. Kumar, MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System.
URL citeseer.ist.psu.edu/karypis95metis.html
- [52] T. Tannenbaum, M. Litzkow, Checkpointing and migration of UNIX processes in the Condor distributed processing system, Dr Dobbs Journal.
- [53] X. Dimitropoulos, D. Krioukov, G. Riley, Revisiting Internet AS-level topology discovery, in: Proceedings of 6th Passive and Active Measurement Workshop (PAM'05), 2005.
- [54] H. Chang, R. Govindan, S. Jamin, S. J. Shenker, W. Willinger, Towards capturing representative AS-level Internet topologies, Computer Networks Journal 44 (2004) 737–755.
- [55] RIPE, <http://www.ripe.net>.
- [56] A traceroute server list, <http://www.traceroute.org>.
- [57] B. Zhang, R. Liu, D. Massey, L. Zhang, Collecting the Internet AS-level topology, Computer Communications Review 35 (1).
- [58] k claffy, T. E. Monk, D. McRobb, Internet tomography, Nature [Http://www.caida.org/tools/measurement/skitter/](http://www.caida.org/tools/measurement/skitter/).
- [59] CAIDA, Macroscopic Topology AS Adjacencies, <http://sk-aslinks.caida.org>.
- [60] N. Spring, R. Mahajan, D. Wetherall, Measuring ISP topologies with Rocketfuel, in: ACM SIGCOMM, 2002.
- [61] R. Govindan, H. Tangmunarunkit, Heuristics for Internet map discovery, in: IEEE INFOCOM 2000, IEEE, Tel Aviv, Israel, 2000, pp. 1371–1380.
URL citeseer.nj.nec.com/govindan00heuristics.html
- [62] A. Lakhina, J. Byers, M. Crovella, P. Xie, Sampling biases in IP topology measurements (2002).
URL citeseer.ist.psu.edu/lakhina03sampling.html

- [63] L. Li, D. Alderson, W. Willinger, J. Doyle, A first-principles approach to understanding the Internet's router-level topology, in: Proceedings of ACM SIGCOMM, 2004.
- [64] X. Dimitropoulos, D. Krioukov, B. Huffaker, k claffy, G. Riley, Inferring AS relationships: Dead end or lively beginning?, in: Proceedings of 4th Workshop on Efficient and Experimental Algorithms (WEA' 05), 2005.
- [65] L. Subramanian, S. Agarwal, J. Rexford, R. H. Katz, Characterizing the Internet hierarchy from multiple vantage points, in: Proc. of IEEE INFOCOM 2002, New York, NY, 2002.
URL citeseer.nj.nec.com/subramanian02characterizing.html
- [66] T. Erlebach, A. Hall, T. Schank, Classifying customer-provider relationships in the Internet, in: Proceedings of the IASTED International Conference on Communications and Computer Networks (CCN), 2002.
- [67] G. D. Battista, M. Patrignani, M. Pizzonia, Computing the types of the relationships between Autonomous Systems, in: IEEE INFOCOM, 2003.
- [68] W. B. Norton, Internet Service Providers and peering, Equinix white paper, <http://citeseer.ist.psu.edu/norton00internet.html>.
- [69] D. M. Nicol, S. W. Smith, M. Zhao, Efficient Security for BGP Route Announcements, Tech. Rep. TR2003-440, Dartmouth College, Computer Science, Hanover, NH (May 2003).
URL <ftp://ftp.cs.dartmouth.edu/TR/TR2003-440.R2.ps.Z>
- [70] J. Xia, J. Hua, Benchmarking and simulation of BGP processing, not published (Dec 2002).
URL http://www.ecs.umass.edu/ece/wolf/courses/ECE697J/Fall12%002/projects/proj_bgp_rep.pdf
- [71] A. Feldmann, H. Kong, O. Maennel, A. Tudor, Measuring BGP pass-through times, in: Proc. of 5th annual Passive and Active Measurement Workshop, 2004.
- [72] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, C. Diot, Characterization of failures in an IP backbone network, in: IEEE INFOCOM, 2004.