BGP Emulation for Domain-Wide Route Prediction

Nick Feamster
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139
feamster@lcs.mit.edu

Jared Winick
EECS Department
University of Michigan
Ann Arbor, MI 48105
jwinick@eecs.umich.edu

Jennifer Rexford
Internet and Networking Systems
AT&T Labs – Research
Florham Park, NJ 07932
jrex@research.att.com

Abstract—**Network operators modify the configuration of routing protocols to adapt to traffic shifts, equipment failures, and the outlay of new capacity. Operators can reconfigure the Border Gateway Protocol (BGP) to control how traffic flows to neighboring Autonomous Systems (ASes) and how traffic traverses the domain to reach the egress points. Unfortunately, predicting the impact of configuration changes is surprisingly difficult because of the distributed nature of BGP and complex interactions with other routing protocols. This paper presents a BGP emulator that determines how traffic flows through an AS using only a static snapshot of the network configuration and the routes advertised by neighboring ASes. Rather than simulating the details of BGP message passing, the emulator predicts the outcome of the BGP decision process on each router using a centralized algorithm and representative abstractions of the relevant inputs—advertised routes from neighboring ASes, local BGP routing policies, internal BGP configuration, and intradomain routing parameters. To handle the magnitude and diversity of these inputs, our prototype implementation limits computational overhead by exploiting the unique structure of the routing data. We validate and evaluate our prototype using data from AT&T's commercial IP backbone.**

I. Introduction

Network operators modify routing protocol configurations to adapt to the outlay of new capacity, large shifts in offered traffic, or significant routing changes in neighboring domains. To ensure reasonable performance during equipment failures and maintenance activities, operators need to predict the link loads under different variations in the topology and perhaps modify the routing configuration accordingly. Predicting the routes that each router in an Autonomous System (AS) selects for each destination would allow an operator to debug performance and reachability problems. Nevertheless, basic information about Internet routing behavior is surprisingly elusive. For example, because of the distributed nature of the Border Gateway Protocol (BGP) [1] and complex interactions with other routing protocols, a network operator cannot easily determine how traffic would travel through the domain en route to other destinations, even with full knowledge of an AS's topology and router configuration. To avoid costly debugging time and catastrophic mistakes, network operators should be able to make these types of predictions based on an accurate model of the routing protocols.

To solve this problem, we present an emulator that predicts how traffic would travel through an AS, given only a static snapshot of the network configuration and the routes advertised by neighboring domains. Existing emulation techniques focus on Interior Gateway Protocols (IGPs), such as Open Shortest Path First (OSPF), which select shortest paths within an AS based on link weights set by network operators [2]. These emulation models are the foundation for traffic engineering schemes that tune the link weights to the prevailing traffic and topology [3]. However, these models do not capture the effects of BGP, which controls how traffic travels between ASes. In contrast to IGPs, which compute shortest paths based on static link weights, BGP path selection depends on routing policies (configured by network operators) and a complex, asynchronous decision process (implemented by router vendors). These factors introduce considerable challenges to the emulation problem.

This paper presents an abstract model that describes how each BGP-speaking router in an AS selects a best route to each block of destination IP addresses. Previous research on interdomain routing has focused on understanding the *structural* properties of the Internet topology [4, 5] and the *dynamics* of the BGP protocol [6, 7], rather than rigorously modeling BGP path selection at an AS-wide level. Formal models of BGP have been used to analyze protocol convergence [8–11] but do not capture the route selection process *within an AS*. In contrast, our work provides both a precise understanding of how BGP route selection depends on the network state and an efficient way to predict the routing choices made at each router in the domain. Efficiency is crucial, since we envision human operators and automated tools using the emulator as an "inner loop" to explore many possible routing configuration changes.

Emulating BGP at a domain-wide level presents the following challenges:

- *BGP policies have an indirect effect on route selection:* BGP route advertisements contain numerous attributes, and operators have considerable flexibility to specify policies that manipulate these attributes to affect route selection. However, these policies have only an *indirect* influence on the complex multi-stage path selection process at each router.
- *Route selection depends on complex protocol interactions:* Internal BGP (iBGP) configuration affects route propagation within an AS and affects the advertisements that a router receives. The *intra*domain routing configuration also affects BGP's best route selection, since IGP weights dictate the closest network exit point. Finally, the best route at one router may depend on the routes selected by *other* routers in the AS.
- *Route emulation depends on a large volume of input data:* The emulator requires a domain-wide view of the network

state. Thus, it must efficiently process and combine the eBGP-learned routes, the BGP import policies, the iBGP session topology, and the IGP parameters.

To solve these problems, we develop the following set of techniques and tools:

• *Route prediction algorithm:* We present an algorithm that efficiently emulates the BGP decision process from a snapshot of the network state rather than simulating the details of individual BGP messages.
• *Vendor-independent abstractions:* In practice, policy specification and router configuration are expressed in obscure, vendor-specific languages. We provide abstractions for BGP routing policies and iBGP configuration that represent this state in a concise, vendor-independent fashion.
• *Prototype emulation tool:* We built a database-driven route emulation tool based on our algorithms and abstractions. We use routing and configuration data from AT&T's commercial backbone to verify the correctness and performance of the tool. The emulator correctly predicts the outcome of the BGP decision process more than 99% of the time and is efficient enough to be used both by network operators and as an inner loop for an automated optimization engine.

After a brief overview of interdomain routing, we present our emulation algorithms and describe the implementation. We then evaluate the prototype's correctness and efficiency. We conclude with a summary of our results and a discussion of future research directions.

II. Challenges for Domain-Wide BGP Prediction

In this section, we present an overview of BGP and describe how routing policies affect the selection of the best route at each router. We then discuss how a router combines information from multiple routing protocols to construct the forwarding table and highlight how the interaction between the protocols complicates the route emulation problem.

A. BGP Decision Process and Policy Interaction

BGP is a path-vector protocol that constructs a route by successively propagating reachability information. A BGP-speaking router sends an advertisement to notify its neighbor[1] of a new route to the destination prefix and sends a withdrawal to revoke the route when it is no longer available. Each route advertisement includes various attributes, such as the list of the ASes in the path (the *AS path*) and the IP address of the router responsible for the route (the *next hop*). If a router learns routes for a prefix from multiple BGP neighbors, the BGP *decision process* selects a single "best" route. In the simplest case, a router selects the route with the shortest AS path and advertises this path to each of its BGP neighbors. In practice, however, locally-configured import and export policies typically affect the selection and propagation of routes.

[1] Throughout the paper, "neighbor" refers to an eBGP or iBGP adjacency.

1. Highest local preference
2. Lowest AS path length
3. Lowest origin type
4. Lowest MED (with same next-hop AS)
5. eBGP-learned over iBGP-learned
6. Lowest IGP path cost to next hop
7. Lowest router-id of BGP speaker

Table 1. The steps in the BGP decision process

The BGP decision process proceeds in several steps, as summarized in Table 1. Setting the *local preference* attribute influences the outcome most directly. An operator can configure the import policy to assign local preference based on the AS path or other route attributes. For example, suppose that all of the routers have an import policy that assigns a local preference of 100 to all routes. The operator could reduce the load on a congested edge link by assigning a smaller local-preference value for some of the routes associated with that link. An operator could modify the import policy at this router to assign a local preference of 80 for, say, AS paths ending in 65000, while continuing to assign 100 to all remaining routes. This modification would make routes to AS 65000 via this router less attractive to other routers in the AS, which would instead choose another network exit point, thus reducing the load on the congested link. The programmability of the import policies gives operators considerable flexibility in controlling the traffic flow at network exit points [12].

Several other attributes also affect the decision process. The *origin type* identifies how the originating AS learned about the route, where IGP is preferable to EGP (a now-defunct distance-vector protocol), which is preferable to INCOMPLETE. The route advertisement may also include a *multiple exit discriminator* (MED) to encourage the recipient to select a particular egress point for sending traffic to the neighboring AS; this is typically done by advertising different MED values via different BGP sessions between the two ASes. As such, the MED attribute only has meaning in comparing routes learned from the same neighboring AS. However, a router's import policy can override the origin type and MED attributes, so their use typically depends on a mutual understanding between the two neighbors. If the first four steps in Table 1 do not result in a single best route, the router prefers a route learned directly via an eBGP session instead of any iBGP-learned route. Then, the route with the smallest IGP cost is preferred (i.e., hot-potato routing). The final tie-break depends on the router-ids of the BGP speakers who advertised the routes.[2]

B. Routing Protocol Interactions

Ultimately, each router synthesizes information from several routing protocols to construct a forwarding table to map destination prefixes to outgoing links. In large service provider

[2] Some router vendors select the "oldest" advertisement, to favor more stable routes. However, "age-based" tie-breaking introduces non-determinism in the BGP path selection process; as such, this step is often disabled.

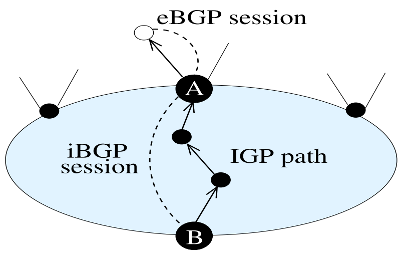

Fig. 1. Interaction between eBGP, iBGP, and IGP in an AS

networks, each routing protocol plays a specific role:

- **eBGP:** Information conveyed via eBGP allows a router A to learn which neighbor to use to direct traffic toward a particular destination prefix, as shown in Figure 1. The edge routers have eBGP sessions to exchange reachability information with neighboring domains. Each eBGP session has an import policy and a router-id and is associated with one or more outgoing links to a next-hop AS. Applying the import policy is a local operation that discards a route or changes its attributes as part of constructing the BGP routing table.
- **iBGP:** Propagating routes via iBGP allows another router B in the same domain to learn that A has a route for the prefix. An iBGP session operates in the same fashion as an eBGP session, with the exception that routes learned from one iBGP neighbor are not advertised to another iBGP neighbor. In addition, iBGP sessions typically do not apply policies that manipulate the BGP attributes of the routes. Rather than having a full mesh of iBGP sessions, a large AS may introduce hierarchy through the use of route reflectors. A route reflector (RR) receives iBGP routes from various neighbors and propagates its best route to its clients. The iBGP sessions form a *signaling graph* [9] connecting the routers in the AS.
- **IGP:** The IGP allows the router B to determine whether A is the closest exit point with a route for the prefix and to identify which of B's outgoing links lie on shortest paths to A. The intradomain topology consists of the routers in the AS and the links between them, where each unidirectional link has an operator-configured weight and (optionally) an area. The path cost for each router pair can be computed directly from a snapshot of the IGP topology and configuration [2].

Despite the separation of roles between these three routing protocols, various protocol interactions complicate modeling the path selection process. First, the iBGP and IGP configurations can interact in subtle ways. A route reflector may select and propagate a different best route than its clients would have chosen with all of the options at their disposal; in extreme cases, this can lead to route oscillation [9, 11]. In addition, a router may forward a packet to another router that has selected a different exit point for the destination prefix; this can lead to unexpected forwarding paths, including loops [9]. Second, the way the BGP decision process handles the MED attribute (comparing routes only if they have the same next-hop AS) makes it difficult to impose a ranking on the BGP routes learned at a router. A route a may be "better" than b, which in turn is better than c, which in turn is "better" than a.[3] Under certain iBGP configurations, the MED attribute can trigger route oscillation within an AS [13]. Ensuring that the emulator captures the complex interaction between the routing protocols is the subject of the next section.

III. Route Prediction Algorithms

In this section, we describe the algorithms that determine the best route at each router in an AS. The algorithms perform route prediction for an entire AS in a single pass by abstracting individual routing protocol messages, routing policy specification, and router configuration. To cope with the complexities highlighted in Section II, our design decomposes the emulator into three independent modules shown in Figure 2.

After describing this decomposition, we present algorithms for computing the set of egress points for each prefix and selecting a single egress point for each router for that prefix. Although the first module in Figure 2 is not especially complicated, the second and third modules require efficient algorithms for computing the best eBGP-learned routes and determining the best route for each router in the domain, respectively. Since the path selection process for each prefix is independent, we focus the discussion on the problem for a single destination prefix.

A. Decomposing the Route Prediction Problem

An instance of the route prediction problem consists of a snapshot of the network configuration and the eBGP-learned routes for a given AS. A solution identifies how traffic from an ingress point to a destination prefix would travel through the AS to a particular egress point to a neighboring domain. To mitigate complexity, we decompose the problem into three independent modules, as shown in Figure 2:

- **Apply import policy on eBGP sessions:** The first module applies the import policy to the routes learned via each eBGP session. These routes, with modified attributes, form the input to the second module.
- **Compute the set of best eBGP routes:** The next module identifies the eBGP-learned routes that could be chosen as the best route by some router in the domain. For each destination prefix, the second module outputs a *set* of eBGP-learned routes that are equally good up to the IGP tie-break in step 6 of the decision process.
- **Compute the best BGP route at each router:** The third module captures how each router selects a best route for each

[3]For example, suppose a router has two routes with the same next-hop AS—a route a learned via iBGP from another router in the domain and an eBGP-learned route b with a larger MED. Suppose also that the router has a route c learned via eBGP from a different neighboring AS, where this session has a larger router-id than the session advertising b. If all three routes have the same local preference, AS path length, and origin type, the decision comes down to steps 4–7 in Table 1. The router prefers a over b (due to MED), b over c (due to router-id), and c over a (due to eBGP vs. iBGP).

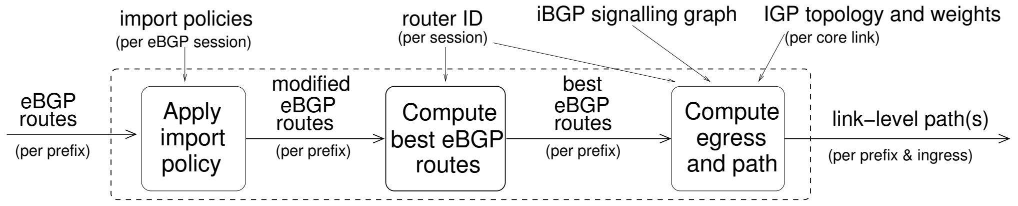

Fig. 2. Decomposing the BGP route emulator into three independent modules

destination prefix from the routes learned via eBGP and iBGP. For each destination prefix and ingress point, this module outputs a network egress point. This module captures the role of iBGP, IGP, and BGP router-id (steps 6 and 7 in Table 1).

The prediction algorithms assume that the eBGP-learned routes are stable; that is, the emulator uses the assumption that the routes seen at the time of a routing table dump are an accurate indication of the routes that would be available in the future. Although BGP updates may cause routes to appear and disappear over time, most routes are stable, and the vast majority of traffic is destined to prefixes with stable eBGP routes [14].

We envision that operators and network architects could couple the modules shown in Figure 2 with additional functionality. For example, a module that mapped the eBGP sessions to the egress links that would carry the traffic could combine the predicted paths with traffic measurement data to predict the load on each link in the network. If a network operator had measurements (or estimates) of the traffic arriving at each ingress point for each destination prefix [15], he could combine this information with the paths predicted by the third module and sum over all of the paths to predict the load on each link in the network. Another module might evaluate the optimality of any particular path assignment from the third module (e.g., in terms of propagation delay, link utilization, etc.) and could search for good configuration and policy changes. As another possibility, import policies and other router configuration inputs could be tested to ensure that sufficient conditions for convergence are satisfied.

B. *Computing the Set of Best eBGP Routes*

The second module starts with the eBGP routes (after modification by the import policies) and produces the set of best eBGP routes for each prefix. This set contains no more than one route to a prefix for each eBGP-speaking router and indicates which egress point that router would use to send packets destined to that particular prefix. To compute the set of best eBGP-learned routes, every router in the AS must learn one or more of the routes in this set. Otherwise, an isolated router might select a route that is less preferable to routes learned at other routers in that AS. Thus, each eBGP-speaking router must have a path in the iBGP signaling graph (perhaps through one or more route reflectors) to every other router in the domain where we want to predict the BGP routing decision. The recommended practice of connecting top-level route reflectors in a “full mesh” of iBGP sessions [16] results in networks that satisfy this constraint.

Given this constraint, the second module seems to have a relatively simple solution: take the set of eBGP-learned routes across all of the routers and identify the best routes by applying the BGP decision process. In the absence of MEDs, this approach would work. However, the way the BGP decision process handles MEDs precludes this approach. Instead, the second module gradually eliminates routes from the set of modified eBGP routes, as follows:

1. **Local-pref, AS path length, and origin type elimination:** Eliminate eBGP-learned routes that do not have the highest local preference. From the remaining routes, eliminate those that do not have the smallest AS path length. Finally, eliminate remaining routes that do not have the smallest origin type.
2. **Local MED elimination:** From the remaining routes, for each router, among routes with the same next-hop AS, eliminate routes that do not have the minimum MED value. At the end of this phase, a router may have two or more routes that are equally good except for the router-id.
3. **Global MED and router-id elimination:** While there are still routers with one or more candidate routes, select any router and consider its locally-best route (the one with the smallest router-id). If this route has a lower MED value and the same next-hop AS as the locally-best route at any other router eliminate this route. Otherwise, eliminate all other routes at this router as well as all other routes with the same next-hop AS and a larger MED value. The remaining routes are the best eBGP routes.

The algorithm is linear in the number of eBGP-learned routes. Previous work formally presents the algorithm and proves its correctness [17].

C. *Computing the Best Route at Each Router*

The third module in Figure 2 determines the best route at each router and computes the path from this router to the egress point. On the surface, this problem has a relatively simple solution: take the set of best eBGP-learned routes from the second module and select the “closest” egress router (in terms of IGP cost), breaking ties based on the BGP router-ids. However, this approach requires that each router learns the complete set of eBGP routes and that no other routers along the path to the selected egress point choose a differ-

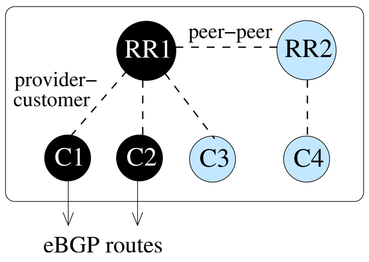

Fig. 3. Example iBGP signaling graph highlighting two-phase algorithm

ent egress point. The use of route reflectors (RRs) can result in network configurations that violate both of these assumptions. However, previous work establishes sufficient conditions to ensure that iBGP configuration is not susceptible to these problems [9]. These conditions, which we discuss below, should be checked before the third module attempts to compute the best route.

Even when these conditions hold, one router's routing decisions can affect the routing options at other routers. To account for these dependencies, our algorithm walks through the iBGP signaling graph to make decisions at each router and propagate the effects of these decisions to other routers. Following the terminology from previous work [9], we refer to an iBGP session from a router to its RR as "customer-provider" and a session between two RRs at the same level in the iBGP hierarchy as "peer-peer". Figure 3 shows an example with a single peer-peer session between two route reflectors and four customer-provider sessions. The sufficient conditions for correct iBGP configuration [9] state that a router must prefer routes learned via iBGP "customers" over routes learned from iBGP "peers" and "providers" (i.e., the router must have a lower IGP path cost to its iBGP customers) and the customer-provider graph must be acyclic (e.g., if a is a route reflector for b, and b is a route reflector for c, then c is *not* a route reflector for a).

If these constraints are satisfied, the algorithm can emulate the exchange of iBGP messages in two phases by considering the routing decisions at the routers in the following order:

1. **Customers first:** Consider the routers in an order that conforms to the partial order in the *customer-to-provider* signaling graph. In this phase, a router selects a customer route, if one is available, only after its customer has selected a route. For example, in Figure 3, C1 and C2 have a best eBGP-learned route and route-reflector RR1 is considered; RR1 selects one of these two routes, based on the IGP cost and the router-ids. C3, C4, and RR2 cannot be considered since no customer route is available.

2. **Providers next:** Consider the remaining routers in an order that conforms to the partial order in the *provider-to-customer* signaling graph. In this phase, routers that did not select a customer route choose a peer or provider route as they become available. For example, either RR2 or C3 can be considered first, since all upstream iBGP neighbors have already selected a route. Once RR2 has been considered, C4 can be considered.

The algorithm can consider the routers without backtracking as long as the network configuration obeys the sufficient condition that guarantees convergence to a unique solution. Since routes from customers are preferable to routes from peers or providers, any router that selects a route in the first phase would never select a different route based on any decision made in the second phase. Within a phase, the activation order emulates the percolation of information up and then down the iBGP hierarchy. The algorithm follows the same approach as the constructive proof of a theorem from previous work that states sufficient conditions for stable interdomain routing at the AS level (Theorem 5.1 from [10]); this theorem also underlies the derivation of the sufficient conditions for iBGP configuration [9]. Once each router has a best egress point, the IGP configuration determines the forwarding path from each router to its egress point; the sufficient conditions for correct iBGP configuration ensures that no router along the path "deflects" the traffic toward a different egress point [9].

D. Checking for Sufficient Conditions

Ultimately, certain network configurations thwart our emulation algorithms. This is a fundamental problem underlying BGP—some configurations do not converge, and determining whether a system converges to a unique solution is computationally intractable [8]. Our emulator assumes that the configuration obeys the *sufficient* conditions for convergence to a unique solution free of IGP path deflections. A route emulation tool should incorporate checks to ensure that these conditions are satisfied to prevent to potential routing anomalies and sources of nondeterminism.

The emulation tool could potentially invoke more complex algorithms to analyze these network configurations (e.g., to report all possible solutions to the routing problem). However, these algorithms would, necessarily, have a longer running time that would make them difficult to use as an "inner loop" for exploring control options. Our implementation of the emulator does not perform these tasks.

IV. Prototype Implementation

In this section, we describe the implementation of the BGP emulator. First, we discuss the advantages of using a database and present an overview of the data model. Next, we describe the operation of each module, including how we populate the tables from external measurement data. For each module, we also discuss how the algorithm avoids duplicate computations if multiple prefixes have the same characteristics or an operator experiments with minor changes to import policies.

A. Database-Driven Implementation

A database provides several advantages for implementing the prototype. On a given day, a large backbone provider such

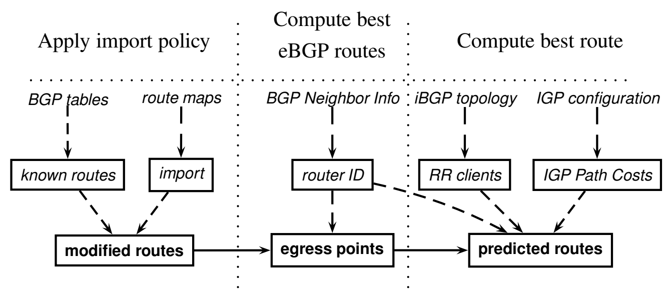

Fig. 4. Dependencies across raw and derived tables in the emulator. In practice, network operators might collect raw inputs on a daily basis.

Table	*Description*
	Module 1: Applying import policy
known routes	router, neighbor, prefix, AS path, AS path length, MED, origin
import	router, neighbor, AS regexp, localpref, MED, origin
modified routes	router, neighbor, prefix, AS path length, localpref, MED, origin
	Module 2: Computing the set of best eBGP routes
router ID	router, neighbor, router-id
egress points	prefix, egress router, neighbor
	Module 3: Computing the best route at each router
RR clients	client, route reflector, cluster-id
IGP path costs	router1, router2, cost
predicted routes	prefix, ingress router, egress router, neighbor

Table 2. Database tables (input data in *italics* and derived tables in **bold**)

as AT&T may have over one million distinct eBGP-learned routes. Any proposed change to the import policies would likely affect only a small subset of those routes; the database interface provides a mechanism for quickly locating the affected routes. Second, the various modules in the emulator must access the data by prefix, BGP session, import policy, router, or some other property, which a database makes particularly easy. Third, the emulation tool should support experimentation with different policies, which means that the tool should be able to quickly restore the default configuration; a database also makes this particularly easy. Finally, database tables provide a clean abstraction between the different modules of the prototype: modules only interact with one another via intermediate results as expressed in the database tables.

Table 2 shows the tables that are used in the tool. Figure 4 summarizes the input data used to construct the abstractions, as well as the intermediate tables generated by each module. Each module computes a single table that is used by the subsequent module. The first module loads the eBGP-learned routes from BGP routing tables and the import policies from the route maps configured on the routers. The module applies the import policies to manipulate the attributes of the routes and generates a new table that serves as an input to the second module. The second module loads a table with the router-id for each session and applies the algorithm from Section III-B to generate the set of best eBGP learned routes, which serves as an input to the third module. The third module loads the iBGP signaling graph and IGP path costs from the configuration data and uses the algorithm from Section III-C to determine the best BGP route for each prefix at each router.

B. Applying Import Policy

The first module applies the import policies to modify the attributes of the eBGP-learned routes.

B.1 Inputs: BGP routing tables and router configuration files

The first module loads the eBGP-learned routes from the BGP routing tables, ignoring any attributes such as local preference that would be assigned by the import policy. Each row in the *known routes* table contains one route, consisting of the prefix, the router at which that eBGP route was learned, the eBGP neighbor, and a column for each route attribute. For a network with a large number of eBGP-learned routes, loading the *known routes* table is very time-consuming. To expedite the process, we load the tables from different routers in parallel. Because many routes have the same AS path, the AS path in the *known routes* table is actually represented as a pointer into a table of unique AS paths. This technique speeds queries for routes based on certain AS path regular expressions, since the table of distinct AS paths is much smaller than the *known routes* table.

The import policy module parses the router configuration files into its corresponding abstract format. The *import* table contains an abstract import policy representation. An eBGP session is uniquely identified by the router name and the next-hop IP address (the remote end-point of the eBGP session). The import policy for each session consists of one or more clauses, represented in the Cisco router configuration files as route-maps. Each clause includes a match expression to target a specific subset of the routes and an action to set or manipulate certain attributes of these routes. In the prototype, we model the common practice of applying a regular expression to the AS path attribute to match particular routes, followed by an action such as setting the local preference, MED, or origin type attributes. The ordering of the clauses is implicit in the order of the rows in the *import* table.

B.2 Algorithm: Applying import policy to known routes

Import policy application applies the rules specified in the *import* table to the *known routes* table. For each row in the *import* table, the algorithm finds all routes in *known routes* that match the BGP session (i.e., router and neighbor) and the AS-path regular expression or prefix to match and alters the route attributes accordingly, unless the route matched an earlier clause from the same import policy. The results are stored in a separate **modified routes** table, rather than overwriting the *known routes* table. This enables an operator to revert back to the initial known routes in order to experiment with various changes to the import policies. In contrast to the *known routes* table, the **modified routes** table includes the local preference attribute (set by the import policy) and excludes the AS path attribute (since only the length of the AS path affects the BGP decision process).

C. *Computing the Set of Best eBGP Routes*

The second module determines set the of best eBGP routes (the **egress points** table) from the attributes of the **modified routes** and the router-id associated with each eBGP session.

C.1 Inputs: Obtaining per-session router-id

The last stage of the BGP decision process depends on the router-id associated with the BGP session announcing the route. On a Cisco router, the `show bgp neighbor` command provides the router-id (and various other information) for each BGP session at the router. The emulator parses the output of this command to load a *router ID* table that contains the router name, neighbor IP address, and router-id for each session. The second module uses only the router-ids of the *eBGP* sessions; the third module uses the router-ids for the *iBGP* sessions.

C.2 Algorithm: Computing best routes at exit points

To capitalize on the database, we implement the algorithm from Section III-B through successive restrictions on an SQL query, as shown in Figure 5. For example, the module first determines the maximum local preference across all eBGP-learned routes and adds a restriction to the "select" statement to select only the routes with this local-preference value; this process repeats for the AS path length and origin type. In the absence of MEDs, this form of successive refinement would be sufficient to compute the **egress points** table. The remainder of the algorithm handles the second and third steps outlined in Section III-B. The second step is implemented by selectively refining the SQL query based on the MED attribute, albeit locally at each router based on the next-hop AS. The output of this step is a table of candidates for the best eBGP routes. The third step removes entries from this table by considering a best route at each edge router and eliminating routes with the same next-hop AS that have a higher MED value, as well as all remaining eBGP-learned routes at this router.

Because of the large number of prefixes and eBGP sessions, determining the set of best routes is a computationally intensive task. To reduce the overhead, we exploit the fact that many prefixes are advertised in exactly the same fashion across all eBGP sessions with neighboring ASes [12, 18]. This typically happens when a single institution announces several prefixes from a single location, or a single peer advertises various prefixes with the same AS path length. Because of this commonality across prefixes, the emulator can compute the egress points once for each group of prefixes with a common set of **modified routes**, rather than separately per prefix. To further reduce the overhead, the emulator can avoid recomputation when an operator explores the effects of a small change in import policy. In particular, the emulator can determine which routes are affected by the policy change and recompute the best routes only for these prefixes.

```
PROCEDURE PREDICT-EGRESS-POINTS(PFX, MASK )
  // Initialize restrictions on query.
  RESTRICTIONS = prefix=PFX and masklength=MASK

  // Get the maximum local-pref value among these routes and add it to the set of restrictions.
  MAX_LP ← select max(localpref) from modified_routes
      where RESTRICTIONS
  RESTRICTIONS += and localpref=MAX_LP

  // Get the minimum path length from this set of routes.
  MIN_PL ← min(pathlength) from modified_routes
      where RESTRICTIONS
  RESTRICTIONS += and pathlength=MIN_PL

  // Get the minimum origin type from this set of routes.
  MIN_ORIG ← min(origin) from modified_routes
      where RESTRICTIONS
  RESTRICTIONS += and origin=MIN_ORIG

  ROUTERS ← select distinct router from modified_routes
      where RESTRICTIONS
  RESTRICTIONS += and (

  // Per router, per next-hop AS, figure out the minimum MED values.
  foreach ROUTER in ROUTERS
    NEXTHOP-ASES ← select distinct nexthop-as from modified_routes
      where router=ROUTER
      foreach NEXTHOP in NEXTHOP-ASES
        MIN_MED$_{router,nexthop}$ ← select min(med) from modified_routes
          where router=ROUTER and nexthop-as=NEXTHOP
        RESTRICTIONS += (router=ROUTER and nexthop-as=NEXTHOP
          and med=MIN_MED$_{router,nexthop}$) or
  RESTRICTIONS += )

  // Get the set of candidate best eBGP routes.
  EGRESS_CANDIDATES ← select route, attributes where RESTRICTIONS
  foreach ROUTER in ROUTERS
    // Delete the locally-best routes not better than locally-best routes at other routers.
    do
      NEXTHOP-AS ← select nexthop-as from EGRESS_CANDIDATES
        where RESTRICTIONS and router=ROUTER
      (MIN_MED) ← select min(med) from EGRESS_CANDIDATES
        where RESTRICTIONS and nexthop-as=NEXTHOP-AS
        and router!=ROUTER
      NUM_DELETED ← delete from EGRESS_CANDIDATES
        where router=ROUTER and
        nexthop-AS=NEXTHOP-AS and med > MIN_MED
    while NUM_DELETED > 0

    // Eliminate globally based on MED.
    (MAX_LP$_{router}$, ...) ← BEST-PATH-ATTRS(PREFIX,MASK, ROUTER)
    delete from EGRESS_CANDIDATES
      where (nexthop-as=NEXTHOP-BEST and MIN_MED$_{router}$ < med) or
        (MIN_ROUTER-ID$_{router}$ < routerid and router=ROUTER)

  egress_points ← EGRESS_CANDIDATES
```

Fig. 5. Emulating the BGP decision process at network exit points.

D. *Computing the Best Route at Each Router*

The third module of the emulator captures the effects of the IGP and iBGP configuration on the selection of a specific egress point for each router.

D.1 Inputs: IGP and iBGP Configuration

Two tables capture the IGP and iBGP configuration details that affect the BGP decision process. The *RR clients* table represents the iBGP topology, listing all "customer-provider" relationships between routers. This information defines the signaling graph for a network and allows the emulator to proceed through the two-phase algorithm in Section III-C. The iBGP topology is derived from router configuration data. Each end of a session consists of a router configured to establish an iBGP session to a remote IP address on another router. By combining this information across configuration files, we can identify each pair of routers that has an iBGP session.

The IGP parameters are also derived from the configuration data. From each configuration file, we can identify which interfaces at the router participate in the IGP and the setting of the IGP weight and (optionally) area. Combining information across the configuration files allows the emulator to identify the links between routers [19]. As discussed in Section III-C, the third module in Figure 2 should use these parameters to compute the (shortest-path) cost between each pair of routers, as well as the sequence of links along this path. In the interest of simplicity, the prototype implementation reads an *IGP path costs* table that lists the cost of the shortest path between each pair of routers, as computed by a separate IGP route emulation tool [2]. As such, the prototype currently computes only the chosen egress point, leaving the computation of the IGP path to the existing IGP tool.

D.2 Algorithm: Determining the best route

The third module computes the final solution to route prediction problem. That is, for each destination prefix, the module determines how each ingress router would select a single element of the **egress points** table. The selection of a particular egress point depends on the iBGP configuration and the IGP cost to reach the egress *routers*, rather than the details of the specific eBGP sessions in the egress set. As such, the module first queries to **egress points** to determine the set of egress *routers* associated with each prefix. Then, the module reads the *RR clients*, *IGP path costs* tables, and the router-ids of the iBGP sessions in the *router ID* table and apply the algorithm from Section III-C. Due to the complex interaction between the iBGP and IGP configuration, we do not implement the algorithm as a sequence of SQL queries.

For a particular ingress router, the result of the algorithm depends only on the set of routers in the egress set. In theory, the number of distinct sets of egress routers could be quite large (2^n possibilities for a network with n egress routers). However, in practice many prefixes have the same set of egress routers. This typically happens because a single neighbor AS may advertise many prefixes via all of its peering sessions. Our prototype exploits this observation by caching the best egress router for a given ingress router and set of egress routers. This allows the emulator to avoid reapplying the algorithm from Section III-C repeatedly across all prefixes with the same set of egress routers. As discussed later in Section VI, we achieve significant performance gains from this optimization due to a very large cache hit rate.

V. Validation

In this section, we describe how we validated the route prediction algorithms and prototype implementation using data from AT&T's commercial IP backbone. After an overview of our validation techniques, we present the results for each of the three modules of the prototype implementation.

A. Validation Techniques

Ensuring that the emulator produces correct answers is extremely important, since we envision operators using the tool to guide the changes to the configuration of the operational network. Validation is challenging due to the difficulty of creating a diverse set of network configurations using complete routing protocol implementations on production routers. Network simulators do not capture the full details of the standard routing protocols. In addition, we may be unaware of vendor-specific variations that could affect the accuracy of our results. As such, we test our prototype on a large operational network—AT&T's commercial IP backbone. Since we cannot make arbitrary changes to the network topology or routing configuration, we instead focus on individual snapshots derived from daily dumps of the router configuration files, BGP routing tables, and BGP neighbor information.

To isolate the sources of inaccuracy, we focus on each module independently. The validation of the module in question assumes perfect inputs from all previous modules. At the end of the section, we present statistics from an end-to-end validation of the emulator, where we allow the errors to propagate. For each module, we compare our results to BGP tables from the operational network and present a breakdown of any mismatches we encounter. Where possible, we trace these mismatches to inconsistencies in the data sets, due to differences in the times when the data were collected. This problem is unavoidable since process of "dumping" the network state occurs over a period of several hours, as polling engines contact each router and download large quantities of data. Inevitably, the network state will change during this period. Fortunately, we find that these inconsistencies are infrequent and do not significantly influence the accuracy of the emulation.

The analysis presented in this paper focuses on a snapshot of the network state of the AT&T backbone from early morning (EST) on February 4, 2003. We focus on the eBGP routes learned at the peering points that connect AT&T to other large providers. That is, we exclude the routes learned from customers and instead focus on the routing of outbound traffic to the rest of the Internet. The initial BGP routing data consists of 1,673,780 eBGP-learned routes for 92,348 prefixes with 45,922 distinct AS paths. We apply the tool to these inputs and check whether the emulator produces the same answers that the operational routers selected. In addition to collecting BGP routing tables from the peering routers (where the eBGP

	Policy Change	*Special Case*	*Total Mispredictions*
AS Paths	3	9	12/45922 (0.026%)
Routes	36	277	313/1673780 (0.019%)

Table 3. Summary of errors in applying import policy. Most of the errors resulted from the fact that the prototype implementation does not currently handle a special case in route map configuration.

	Different	*Missing*	*Total*
AS Paths	66	187	253/45922 (0.551%)
Prefixes	120	483	603/92348 (0.653%)

Table 4. Summary of mismatches in predicting the set of best eBGP routes at network exit points. The table shows the number of best eBGP route predictions that did not agree with the route chosen by the corresponding route reflector.

routes are learned), we also collect BGP tables from several other routers in the network to verify the results.

B. Applying Import Policy

To verify that the first module correctly emulates the application of import policy, we need only compare the route attributes (i.e., local preference, MED, etc.) in the **modified routes** table to the actual BGP routing tables. The **modified routes** table contains the routes with attributes modified by applying the import policies specified in the *import* table to the initial *known routes* table. Because the prototype uses routing tables to approximate the actual routes received at each router in the domain, we cannot determine what routes were discarded by the import policy. Thus, our prototype cannot emulate the filtering policies specified by import policies. Nevertheless, the prototype is useful for determining the effects of import policy configurations that set or manipulate the attributes of routes (e.g., for traffic engineering purposes).

We compare the route attributes between the *known routes* table and the *modified routes* table for all 1,673,780 eBGP-learned routes with 45,922 distinct AS paths. Table 3 summarizes the results of our validation. The emulator's results matched the route attributes seen in the BGP tables for all but 313 eBGP-learned routes on 12 distinct AS paths. We observed 36 attribute mismatches over 3 AS paths, which can likely be attributed to actual policy changes, since the routing tables and the configuration files were captured at slightly different times of day; we verified this conclusion by inspecting the configuration data for the next day. The remaining mismatches involved 9 unique AS paths because the prototype did not handle a complex configuration scenario permitted on Cisco routers. This accounted for 277 of the 313 route mismatches. Overall, the first module produced successful results for more than 99.97% of the cases.

C. Computing the Set of Best eBGP Routes

To validate the computation of the set of best eBGP routes, we compare the results of the second module with the best routes selected at each top-level route reflector in the operational network. To verify that the module makes the correct predictions for best routes at eBGP routers, we check that the path chosen by a particular eBGP-speaking router appears in the routing table of the corresponding route reflectors. These routes match the vast majority of the time. However, in a few cases, the two routers had different routes (i.e., with different AS paths), even though one router apparently learned the route directly from the other; these results are summarized in the "Different" column in Table 4. The "Missing" column highlights cases where the RR did not have *any* route for that prefix. Timing inconsistencies can explain both scenarios, which together account for just over 0.5% of the cases.

To verify that the module does not incorrectly exclude routes from the set of best eBGP routes, we check that, for each prefix, the best route at each route reflector appears in the set of best eBGP routes as computed by the emulator.[4] In other words, we consider cases where an RR's best route would have directed traffic towards some egress router that was not contained in the set of best eBGP routes. Only 0.38% of best routes at RRs for 1.3% of prefixes fell into this category. Examination of these anomalies suggests that routing dynamics can explain these inconsistencies as well. Through manual inspection, we found that, in many cases, the best route at the RR was clearly worse than the routes in the set of best eBGP routes (e.g., the RR's best route had the same local preference but a higher AS path length). Often, the corresponding egress router's BGP table had a different route (e.g., with a higher local preference, shorter AS path, etc.), consistent with the "Different" case in Table 4.

D. Computing the Best Route at Each Router

To verify that our emulator accurately predicts the best egress router, we examined the best routes in BGP tables for other routers in the network. Namely, we compared the (destination prefix, next-hop) pair from the routing table with the results in the **predicted routes** table entry for that router. The analysis focused on two access routers that connect directly to customers in different geographic locations. We also analyzed two route reflectors in order to evaluate the way our algorithm traverses the iBGP signaling graph. The best route matched our prediction for 99.5-99.7% of the cases, as summarized in Table 5. The small number of mismatches fall into one of four categories:

- Case 1: The destination prefix at the ingress router does not appear in the **known routes** table, causing the emulator to predict an empty egress set.
- Case 2: The route seen at the ingress router does not appear in the **modified routes** table and, as such, does not appear in the egress set.
- Case 3: The route seen at the ingress router does appear in the **modified routes** table but does not appear in the **egress**

[4]The reverse is not necessarily true. An egress point may have a larger IGP path cost to each of the top-level RRs for certain sets of eBGP routes.

Router	*Case 1*	*Case 2*	*Case 3*	*Case 4*	*Total Mispredictions*
RR1	1	33	326	22	382/82536 (0.463%)
RR2	5	33	187	5	230/81898 (0.281%)
AR1	8	38	180	5	231/81902 (0.281%)
AR2	7	151	170	34	362/70577 (0.513%)

Table 5. Summary of errors in predicting the best egress router (third module). Shown are the number of predictions that do not correspond to the best egress router from the BGP routing table. The 603 prefixes that were incorrectly predicted by the second module are excluded.

Router	*Case 1*	*Case 2*	*Case 3*	*Case 4*	*Total Mispredictions*
RR1	1	33	463	56	553/83139 (0.665%)
RR2	5	33	316	41	395/82501 (0.478%)
AR1	8	38	309	40	395/82505 (0.478%)
AR2	7	151	283	68	509/71180 (0.715%)

Table 6. Summary of errors for end-to-end validation.

points table.

- Case 4: The misprediction has no obvious explanation.

Cases 1 and 2 stemmed from timing inconsistencies, where the route seen at the ingress router was not available at the egress router when the routing table was dumped. Timing inconsistencies also explain Case 3, where the ingress router has a route that (while it exists) is no longer one of the best eBGP-learned routes (say, due to the availability of other, better routes). The unexplained mismatches account for less than 0.05% of the cases.

E. End-to-End Validation

We perform an end-to-end validation to study the effect of error propagation on the best routes ultimately predicted by the prototype. We compared the prototype's prediction with the same four routing tables used for the validation of the third module, with the exception that the input included the errors from the first two modules. At these four routers, the emulator predicted the correct routes for 99.4% of all prefixes. Table 6 summarizes the results of the end-to-end validation.

We hypothesized that the majority of the 0.6% of mispredicted routes could be explained by the dynamics of the input data. A modification to the import policy could have changed the choice of best route between the time the two routing tables were captured. A more likely explanation is that the inconsistencies were caused by routing dynamics that caused the temporary appearance or disappearance of a route. For example, if the best route at an eBGP-speaking router were temporarily withdrawn at the time that the route reflector table was captured, inconsistencies between routing tables could arise.

To evaluate our hypothesis, we analyzed a feed of iBGP update messages collected from the AT&T backbone on the same day. For the prefixes with incorrect route predictions, 45% experienced a BGP update during the data collection period at the same router where the apparent mismatch occurred, and 83% of the prefixes experienced an update at some router in the AS during this period.

Our analysis suggests that most of the prediction errors result from changes in the input. Since most prefixes whose routes change frequently do not receive much traffic [14], these inconsistencies typically would not have a significant effect on the emulator's ability to predict traffic flow. Ideally, the emulator would receive a real-time stream of all of the eBGP-learned routes. However, this is not currently possible because commercial routers only support either (1) dumping the entire BGP table (which contains all of the routes after import processing, but imposes a load on the router and provides only a static view) or (2) having a BGP session to a monitor (which provides a real-time view of only the current best routes after import processing).

VI. Performance Evaluation

In this section, we evaluate the performance of each module. All of our performance tests were conducted on a Sun Fire 15000 server with 192 GB of RAM and 48 900 MHz Ultrasparc-III Copper processors. Because this is a time-shared machine, we ran each of our experiments several times to gain confidence in the timing consistency. Except where noted, the prototype used only two processors (one for the database and one for the emulator itself).

We evaluated running times for data sets of various sizes and scenarios to demonstrate that the emulator is efficient enough to be used in practice, even on a large AS such as AT&T's commercial IP backbone. Most ASes have fewer BGP sessions, peers, and routers, and perhaps fewer prefixes. To quantify the overhead of running the emulator for a small number of sessions, we run the emulator on scenarios with just one or two eBGP sessions. For the one-session experiment, we select one of the eBGP sessions responsible for the largest number of routes in the AT&T network. For the two-session experiment, we include an eBGP session at the same router of comparable size, but with a different neighboring AS. Because we envision that network operators would use the emulator to experiment with the effects of small changes in configuration or topology, we also perform experiments that evaluate the emulator's effectiveness in evaluating incremental changes.

While our evaluation is preliminary, it demonstrates the efficiency of the BGP emulator, especially in the common case of incremental changes.

A. Applying import policy

Table 7 summarizes the total running time for the first module, which applies the import policies to the eBGP-learned routes. This process has four separate steps: (1) parsing and loading the routing tables, (2) parsing and loading the import policies, (3) building the database indexes, and (4) applying the import policies to the eBGP updates. The prototype can parallelize the first two steps by router, since the tables and configuration for each router can be parsed and loaded independently. When the prototype loads these inputs for a partic-

Task	*Initial (sec)*	*Steady-state (sec)*
parse/load routing tables	521 (42.13%)	—
parse/load import policy	0.78 (0.06%)	0.046 (4.64%)
build database indexes	257 (20.78%)	—
apply configuration	458 (37.03%)	0.942 (95.36%)
Total	1236.78	0.988

Table 7. Running time for the first module in the emulator. After the initial setup phase, propagating the effects of a routing policy change takes only about 1 second for the common case.

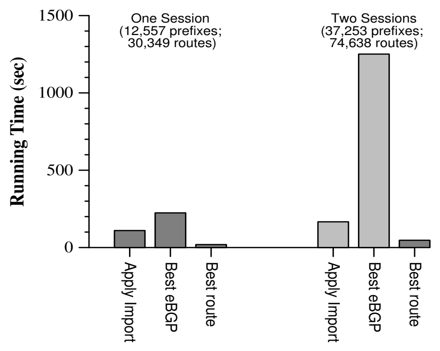

Fig. 6. Performance of the emulator for a smaller number of eBGP sessions

ular day's data, the module must perform all four steps. However, once the routing tables and import policies have been parsed and the indexes have been built, predicting the effects of one configuration change is relatively painless: one router configuration file must be re-parsed, and the new import policy must be applied to the set of routes. Performing these operations for a single import policy change takes less than one second on average.

Figure 6 shows that the initial loading phase performs considerably faster in an AS with only one or two eBGP sessions (common cases for smaller networks). In these cases, performing all tasks associated with the first module takes about 109 seconds and 166 seconds, respectively. The running time is considerably less because there are much fewer routes to parse, from a single routing table.

B. Computing the Set of Best eBGP Routes

Table 8 summarizes the running times for the second module, which computes the set of best eBGP routes. The total running time for 92,419 prefixes was about four and a half hours, or roughly 5.3 prefixes per second. Instead of computing the best routes for 92,419 prefixes independently, the module can avoid duplicate computation when multiple prefixes are advertised in exactly the same way.

Clustering prefixes that have a common set of route attributes[5] takes 813 seconds. Performing route prediction with caching takes 11,220 seconds, or about 8.2 prefixes per second on average. Note that the optimized algorithm takes about 0.58 seconds per group of equivalent prefixes (or 0.12 seconds per prefix) to compute a prediction, whereas the original algorithm required 0.19 seconds per prediction. This apparent average slowdown occurs because checking the cache requires a database query to associate each prefix with a prefix group; since many groups contain multiple prefixes, this requires multiple loop iterations. Nevertheless, caching the results for prefixes that share a common set of **modified routes** does provide an overall speedup of about 31%.

[5]Because the AS path *length* affects the outcome of the second module but the AS path itself does not, the module can cluster prefixes that have different AS paths but the same AS path length. This optimization further reduces the number of predictions that the second module must perform.

	Predictions	*Time (sec)*	*Prefixes/sec*
Without caching	92419	17537	5.27
With caching	8091	12033	8.23

Table 8. Running times for the second module, which computes the set of best eBGP routes. Recognizing that prefixes can be grouped according to how they are advertised produces a significant speedup.

	Hits	*Misses*	*Time (sec)*	*prefixes/sec*
Without caching	—	—	390	211.6
With caching	82245	290	245	336.9

Table 9. Running times and cache performance for the third module, which computes the best egress router. The results are from the validation of RR1, where predictions are made for 82,535 prefixes. This is one less than the 82,536 prefixes that we validated, as one prefix had an empty egress set (a Case 1 error for which we could not make a prediction).

Figure 6 shows the running times for the second module with one and two eBGP sessions and caching enabled. The speedup for a small number of sessions is not as dramatic as for other modules, because, even for a small number of prefixes, there is not a linear reduction in the number of prefixes and routes. Nevertheless, the running times are considerably smaller than for a network as large as AT&T's.

C. Computing the Best Route at Each Router

Table 9 shows the running time and cache performance for the third module. The results come from the validation test of RR1 from Table 5. The performance of the third module is greatly improved by exploiting the nature of routing table data. For an ingress router, we only need to calculate the best egress router for a given egress set once. Without caching, it takes 390 seconds to predict the best egress router for the 82,535 prefixes at this router, or about 212 prefixes per second. When the prototype caches the computation of the best egress router, the running time is 245 seconds, or about 337 prefixes per second. With caching enabled, the algorithm to compute the best egress router only needs to run 290 times, as opposed to 82,535 times, which significantly improves the performance of the third module. Figure 6 shows that module 3's running time is comparatively small: 19 seconds and 47 seconds for one and two eBGP sessions, respectively.

VII. Related Work

Prior work presented an IGP emulator that helps network operators optimize link weights for intradomain traffic engineering [2]. However, this emulator does not model changes to BGP routing policies or the effects of iBGP on path selection. Recent work proposes efficient techniques for large-scale parameter optimization for various network protocols, including the tuning of the local preference attribute in BGP [20]. This work is complementary to ours, as the proposed search techniques could use our emulator as the "inner loop". These techniques use the SSFNet simulator [21], which simulates the message-passing details of the routing protocols. Although SSFNet is useful for studying protocol dynamics, direct computation of routing paths is a more efficient mechanism for parameter optimization. In this paper, we presented an abstraction for router import policy; while similar to RPSL [22], our abstractions are specifically designed to capture semantics that are relevant to protocol emulation.

Previous work defined sufficient conditions for router configuration within an AS to guarantee that the routing protocols converge to a stable, deflection-free path assignment [9, 13]. The emulator we have presented assumes a separate mechanism for checking that a network configuration satisfies these conditions. In previous work, we proposed efficient techniques for operators to tune BGP import policies to engineer the flow of traffic [12]; this work assumes the existence of a BGP emulator. The work also described ways to avoid policy changes that could have unpredictable side effects on ingress traffic, which is necessary to ensure that the inputs to the emulator are deterministic and predictable. We previously proposed a high-level architecture for a BGP emulator, including the separation of modules in Figure 2 and the details of the algorithm for the second module [17]; however, we did not explore the algorithm for the third module or present any implementation, validation, or evaluation results.

VIII. Conclusion

We have presented an emulator that accurately and efficiently predicts the outcome of the BGP route selection process in a single AS using only a snapshot of the network configuration and the eBGP-learned routes from neighboring domains. The emulator models the BGP decision process and the complex interactions with other routing protocols. Our experiments demonstrate the emulator's accuracy and efficiency.

Our abstractions provide a clean separation between each aspect of route prediction, as well as an abstract, vendor-independent representation of the relevant parts of the router configuration. We envision that our work could be combined with higher-level mechanisms that spot misconfiguration or check that other constraints are satisfied.

The prototype depends on many inputs including router configuration files, BGP table dumps, and BGP session information for every BGP-speaking router in the AS. In reality, operators may not have access to all of these inputs, and some inputs may be incomplete or out-of-date. Producing approximate results in the absence of complete information is a promising area for future work.

Acknowledgments

We would like to thank Joel Gottlieb, Tim Griffin, and Carsten Lund for their help in using the configuration data, and Glenn Fowler for his tool for parsing the routing tables. We also thank Magdalena Balazinska, Greg Harfst, and Stan Rost for their comments on a draft of this paper.

References

[1] Y. Rekhter and T. Li, "A Border Gateway Protocol." RFC 1771, March 1995.

[2] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, and J. Rexford, "NetScope: Traffic engineering for IP networks," *IEEE Network Magazine*, pp. 11–19, March 2000.

[3] B. Fortz, J. Rexford, and M. Thorup, "Traffic engineering with traditional IP routing protocols," *IEEE Communication Magazine*, October 2002.

[4] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the Internet topology," in *Proc. ACM SIGCOMM*, pp. 251–262, August 1999.

[5] N. Spring, R. Mahajan, and D. Wetheral, "Measuring ISP topologies with Rocket-Fuel," in *Proc. ACM SIGCOMM*, August 2002.

[6] C. Labovitz, R. Malan, and F. Jahanian, "Internet routing stability," *IEEE/ACM Trans. Networking*, vol. 6, pp. 515–528, October 1998.

[7] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian, "Delayed Internet routing convergence," *IEEE/ACM Trans. Networking*, vol. 9, pp. 293–306, June 2001.

[8] T. Griffin, F. Shepherd, and G. Wilfong, "The stable paths problem and interdomain routing," *IEEE/ACM Trans. Networking*, vol. 10, no. 1, pp. 232–243, 2002.

[9] G. Wilfong and T. G. Griffin, "On the correctness of IBGP configuration," in *Proc. ACM SIGCOMM*, August 2002.

[10] L. Gao and J. Rexford, "Stable Internet routing without global coordination," *IEEE/ACM Trans. Networking*, vol. 9, pp. 681–692, December 2001.

[11] A. Basu, A. Rasala, C.-H. L. Ong, F. B. Shepherd, and G. Wilfong, "Route oscillations in I-BGP with route reflection," in *Proc. ACM SIGCOMM*, August 2002.

[12] N. Feamster, J. Borkenhagen, and J. Rexford, "Techniques for interdomain traffic engineering." *In submission, 2002.* `http://www.research.att.com/~jrex/papers/bgpte.ps`.

[13] T. G. Griffin and G. Wilfong, "Analysis of the MED oscillation problem in BGP," in *Proc. International Conference on Network Protocols*, November 2002.

[14] J. Rexford, J. Wang, Z. Xiao, and Y. Zhang, "BGP routing stability of popular destinations," in *Proc. Internet Measurement Workshop*, November 2002.

[15] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True, "Deriving traffic demands for operational IP networks: Methodology and experience," *IEEE/ACM Trans. Networking*, vol. 9, June 2001.

[16] S. Halabi and D. McPherson, *Internet Routing Architectures*. Cisco Press, 2001.

[17] N. Feamster and J. Rexford, "Network-wide BGP route prediction for traffic engineering," in *Proc. Workshop on Scalability and Traffic Control in IP Networks, SPIE ITCOM Conference*, August 2002.

[18] D. Andersen, N. Feamster, S. Bauer, and H. Balakrishnan, "Topology Inference from BGP Routing Dynamics," in *Proc. Internet Measurement Workshop*, (Marseille, France), November 2002.

[19] A. Feldmann and J. Rexford, "IP network configuration for intradomain traffic engineering," *IEEE Network Magazine*, pp. 46–57, September/October 2001.

[20] T. Ye, H. T. Kaur, and S. Kalyanaraman, "Large-scale network parameter configuration using an on-line simulation framework." *In submission, 2003.* `http://www.ecse.rpi.edu/Homepages/shivkuma/research/papers/ols-j.pdf`.

[21] "SSFNet." `http://www.ssfnet.org/`, 2003.

[22] C. Alaettinoglu *et al.*, *Routing Policy Specification Language (RPSL).* Internet Engineering Task Force, June 1999. RFC 2622.