Using Forgetful Routing to Control BGP Table Size

Elliott Karpilovsky
Computer Science Department
Princeton University
elliottk@cs.princeton.edu

Jennifer Rexford
Computer Science Department
Princeton University
jrex@cs.princeton.edu

ABSTRACT

Running the Border Gateway Protocol (BGP), the Internet's interdomain routing protocol, consumes a large amount of memory. A BGP-speaking router typically stores one or more routes, each with multiple attributes, for more than 170,000 address blocks, and growing. When the router does not have enough memory to store a new route, it may crash or enter into other unspecified behavior, causing serious disruptions for the data traffic. In this paper, we propose a new mechanism for routers to handle memory limitations without modifying the underlying routing protocol and without negatively affecting convergence delay. Upon running out of memory, the router simply discards information about some alternate routes, and requests a "refresh" from its neighbors later if necessary. We present an optimal offline algorithm for deciding which alternate routes to evict, and explore the trade-off between memory size and refresh overhead using a large BGP message trace. Based on these promising results, we design and evaluate efficient online algorithms that achieve most of the performance benefits. We believe that our scheme can significantly improve the scalability and robustness of IP routers in the future.

Keywords

workload characterization, performance optimization, communication networks

1. INTRODUCTION

The successful delivery of traffic through the Internet depends on the smooth operation of the routing protocols running in and between thousands of Autonomous Systems (ASes). The responsibility for stitching these disparate ASes together into a single, coherent network falls to the Border Gateway Protocol (BGP), the Internet's interdomain routing protocol. BGP enables routers to learn paths through other ASes to reach remote destination address blocks, or prefixes[1]. A router stores the BGP routes it learns for each destination prefix in a routing table, known as a Routing Information Base (RIB), and selects a single "best" route for forwarding data traffic; all other routes are "alternates," used when the primary path becomes unavailable. Upon running out of memory for storing the routing table, today's BGP-speaking routers crash, stop accepting new information, or enter some indeterminate state, leading to serious disruptions in the end-to-end delivery of data traffic [1]. In this paper, we propose and evaluate a new technique for containing the size of a BGP routing table, while remaining backwards compatible with the BGP protocol.

In the remainder of this section, we discuss the memory constraints on routers and the limitations of existing remedies, and introduce the concept of forgetful routing. In Section 2, we present a formalism for describing BGP, providing the background knowledge needed to understand forgetful routing; this leads directly into a discussion of forgetful routing itself. Section 3 analyzes the trade-off between memory savings and refreshes by evaluating an optimal, offline algorithm over a large BGP message trace. Section 4 proposes and evaluates several efficient, online algorithms for deciding which alternate routes to evict when the RIB memory is full. In Section 5 we examine which ASes would most benefit from deploying forgetful routing. We discuss related work in Section 6, and conclude the paper in Section 7.

1.1 Memory Limits and Current Workarounds

Given the relatively low cost of memory, and the fact that even conventional desktop PCs often have hundreds of mega-bytes of main memory, the notion that routers would encounter memory limits may seem surprising. Even taking into account that today's routers must store BGP routes for around 170,000 prefixes, and growing [2, 3], the amount of space needed seems small. For example, a router with 10 neighbors, with each neighbor announcing a route for each prefix, would need approximately 130 mega-bytes of memory (assuming 80 bytes to store routing entry information). So why the concern?

The problem with these back-of-the-envelope calculations is that they're a *lower bound* on memory, and in practice much more memory is needed for various reasons. First, the number of prefixes is sometimes much higher, such as when configuration errors or malicious attacks trigger "route leaks," where a router receives BGP announcements for ad-

[1]A prefix is specified with an IP address and a mask indicating the number of relevant bits, *e.g.*, 122.43.2.1/24 represents all 256 addresses starting with 122.43.2.

dress blocks that are not normally visible. As examples: on April 25th, 1997, AS7007 leaked 23,000 routes, causing enough instability to create massive Internet outages [4]; on October 3rd, 2002, 20% of WorldCom's customers lost connectivity due to a configuration error that "...propagated more route-broadcasts than the affected routers could handle" [5]; on December 24, 2004, AS9121 leaked over 100,000 prefixes [6], *etc.* From 1994 to 2004, there have been more than 60 threads about route leaks on the North American Network Operators' Group (NANOG) mailing list *alone* [7]. Second, a router may learn multiple BGP routes for a prefix, especially as ASes increasingly connect to the Internet in multiple locations for better fault tolerance and more flexible load balancing. Third, operating system upgrades generally provide new features and consume more memory, adding to the problem. Fourth, data structure overhead and other implementation details consume additional space. Considering all these factors, the BGP routing table can grow quite large, up to a gigabyte in size, with the risk that an unexpected route leak may drive the memory requirements significantly higher.

There are many partial solutions to the memory problem, but no panaceas, such as:

- *Adding more memory.* RIB memory is much more expensive than conventional SDRAM, often between one to two orders of magnitude more in price. Moreover, determining how much memory to add is very challenging. Thus, upgrading the memory for every router in a large AS to some "acceptable" level is quite expensive. In other cases, such as routers deployed in a satellite network, adding memory may be impossible.

- *Using secondary storage.* Swapping parts of the RIB to secondary storage may seem appealing, but many routers do not have disk drives; network operators are reluctant to rely on disks, as they have relatively high failure rates [8]. Even when secondary storage is available, excellent virtual-memory techniques need to be used to prevent thrashing.

- *Using compression.* Applying compression techniques to the RIB data may yield some memory savings, at the expense of computational overhead in handling new update messages. Since routers need to respond quickly to routing changes, any compression scheme would need to be simplistic and operate only over local chunks of data, severely constraining any savings.

- *Filtering routes.* Route filters can be installed to discard routes that do not meet certain criteria. In fact, the Regional Internet Registries (RIR) publish guidelines for the maximum prefix lengths for various parts of the IP address space [9]. However, these guidelines are merely suggestions, and many ISPs ignore them. This creates a catch-22, where ASes do not filter based on RIR guidelines since so many ASes violate them, leading ASes to feel little pressure to obey them.

- *Enforcing prefix limits.* By imposing a hard limit on the number of prefixes accepted from each neighbor, and tearing down the BGP session when the number is exceeded, a router can avoid dedicating too much memory to a single neighbor. However, unless extremely conservative limits are imposed, the router remains vulnerable to learning too many routes across all of its neighbors [6].

In sum, existing technologies that control BGP table growth are either only applicable in specialized circumstances or are generally ineffective.

1.2 Practical Constraints on Extending BGP

Unfortunately, devising solutions to BGP's memory problem is not an easy feat. Since BGP is the glue that holds the disparate parts of the Internet together, having a "flag day" to replace BGP with a new protocol is infeasible. Any solution must be incrementally deployable, where one AS can upgrade the software on its routers even if other ASes do not. In addition, upgrading the software needs to offer clear benefits to the early adopters, rather than relying on a large-scale deployment before any memory savings are realized. We argue that a good, practical solution should have the following three properties:

No changes in how the routers select paths. The changes to BGP should not affect which paths the routers select for forwarding data packets. BGP is designed to prevent forwarding loops and unexpected loss of connectivity through its decision process, and support flexible routing policies. Changes to the way routers modify route attributes and select paths could lead to inconsistencies in the routing decisions across the network.

Memory savings for ASes that deploy the solution. Since interdomain routing is tied to the dynamics of businesses, economic incentives drive the deployment of BGP modifications. Upgrading all the routers in an AS is time consuming and expensive. An AS that deploys the new software should see memory savings, even if no other ASes have deployed the solution.

No significant increase in routing convergence delay. The scheme must not significantly affect the time it takes for routing changes to propagate across a network, also known as convergence delay. During convergence, multiple routers recalculate their routing tables in response to a topology or policy change, and transient forwarding loops or blackholes (where packets are lost) may occur. An increase in convergence delay translates into additional time when data traffic may be lost or delayed.

1.3 Our Contribution

With these constraints in mind, we propose a new approach that we dub *forgetful routing.* To avoid exceeding the available memory, a forgetful router can selectively discard one or more alternate routes. If a discarded route is needed sometime in the future, the router requests a "refresh" from the neighbor responsible for announcing the route.

Our solution exploits several important aspects of the BGP routing system:

- *Alternate routes are not needed while the primary route is in use.* This enables our scheme to offer significant memory savings by potentially discarding all alternate routes. If a router has, on average, n routes per prefix, it can reduce its total memory usage by a factor n.

- *Every alternate route is some neighbor's best route.* Thus, every forgotten route is always available for retransmission later. This allows a router to always reconstruct its original routing table when needed and thus select the same best routes as conventional BGP.

- *Every alternate route is always one hop away.* As a result, total convergence delay can only ever increase by the time of a single refresh, no matter the size of the network.

- *BGP's route-refresh feature can be used to trigger a refresh from a neighbor.* BGP's route-refresh capability [10] has been a standard for many years and is already deployed in many large ASes. Although designed for a different purpose, it can be used to ask a router to resend BGP announcements. This significantly lowers the barrier to deploying forgetful routing.

Moreover, our solution does not require any changes to the BGP protocol—only software upgrades on the routers themselves; in addition, one AS could deploy forgetful routing even if other ASes do not. Even in the rare case where none of its neighbors support the route-refresh feature, an AS can use forgetful routing to reduce the amount of memory consumed by internally learned routes (i.e., in internal BGP).

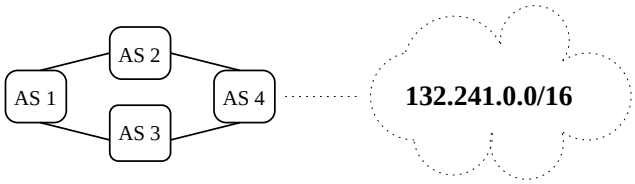
Forgetful routing introduces a trade-off between memory size and refresh overhead, leading to a sort of a "cache replacement" problem. Our goal is to create an efficient eviction algorithm that minimizes the "miss rate," *i.e.*, the likelihood that the best route for a prefix does not already reside in the routing table and will trigger a route refresh. We first present an optimal offline algorithm that assumes perfect knowledge of the future arrivals of BGP update messages. Experiments applying the optimal algorithm to BGP message traces show that forgetful routing can achieve substantial reductions in memory usage, up to a factor of 30, in an idealized scenario. The analysis of the measurement data also provides important insights into the characteristics of BGP update dynamics. We capitalize on these observations in creating two efficient online algorithms that evict the least attractive alternate routes for prefixes that have not changed their best routes for the longest time. Efficient data structures enable a forgetful router to make eviction decisions in constant time, and our experiments show that the online algorithms perform well over actual BGP data, reducing the memory footprint by a factor of 10 with a moderate number of refresh operations.

2. FORGETFUL ROUTING

Describing all aspects of BGP would require an entire book [11], and many of these details are irrelevant to memory management. We first present a simple formalism that captures how BGP-speaking routers handle update messages and select routes; this formalism is used throughout the paper as a reference model. Then, we apply this model to explain forgetful routing and introduce the cache-replacement problem that is the focus of the rest of the paper.

2.1 An Abstract Model of BGP

Initially, a router establishes a BGP session with each neighbor. Each router then shares information about the best routes through update messages. After learning new information from its neighbors, the router checks if better routes exists, and if so, uses them. Upon changing its best route, the router must send an update message to any neighbors that previously received an announcement for the replaced route, indicating that the old route is no longer in use; this withdraw is then followed by an announcement of the new route[2]. If connectivity to a prefix is lost, only a withdraw message is sent.



AS	RIB
AS 1	(132.241.0.0/16, AS3, $1 \rightarrow 3 \rightarrow 4 \rightarrow ...$) (132.241.0.0/16, AS2, $1 \rightarrow 2 \rightarrow 4 \rightarrow ...$)
AS 2	(132.241.0.0/16, AS4, $2 \rightarrow 4 \rightarrow ...$)
AS 3	(132.241.0.0/16, AS4, $3 \rightarrow 4 \rightarrow ...$)
AS 4	(132.241.0.0/16, ..., $4 \rightarrow ...$)

Figure 1: A hypothetical network where AS 1 can route to 132.241.0.0/16. After AS 4 announces a route to the prefix to its neighbors, AS 2 and AS 3 will be able to route to it. They, in turn, generate announcements for AS 1. RIB entries for this prefix are shown using the (p, n, r) notation. Note that AS 1 can always re-derive its RIB entries by issuing route refreshes to its neighbors.

All the information about routes is stored in a Routing Information Base (RIB). A route is a three-tuple (p, n, r) containing a prefix, a neighbor that advertised the prefix, and some route attributes (*e.g.*, the number of hops required to reach the destination, preference values, *etc.*). A RIB is a set of such routes. By default, we assume that for every prefix p, the RIB contains a null route to that destination, *i.e.,* the entry $(p, \emptyset, \emptyset)$, signifying that the router cannot or does not want to route to it.

An *announcement* is an update message of the form:

$$(\text{“}ANNOUNCE'', (p, n, r))$$

that tells the router that neighbor n currently uses route r to reach prefix p. The neighbor n can only advertise its best routes, and must be using the route r to reach p. A *withdraw* is an update message of the form:

$$(\text{“}WITHDRAW'', (p, n, r))$$

indicating that neighbor n can no longer reach prefix p using its previously announced route. The RIB entry corresponding to (p, n, r) is removed. See Figure 1 for an example of how the RIB becomes populated.

Once the RIB becomes populated with entries, the router must decide how to route data to all accessible prefixes. The decision is calculated by using a total ordering[3] over

[2]In practice, an announcement is an implicit withdrawal of the previous route, requiring only one update message.

[3]In practice, some ASes use the Multiple-Exit Discriminator (MED) attribute in a way that prevents the formation of a total ordering [12]. This leads to other problems, such as non-deterministic routing decisions and protocol oscillation. In our work, we assume that MED is not used, or is configured to produce a deterministic outcome. When this assumption does not hold, the simplest solution is to store all routes for a prefix if any route has the MED set.

```
while session exists:
   (m, (p, n, r)) = get_message()
   oldbest = m_best(p, RIB)
   if m == ''ANNOUNCE'':


      RIB = RIB + (p, n, r)
   if m == ''WITHDRAW'':
      RIB = RIB - (p, n, r)
   newbest = m_best(p, RIB)
   if oldbest != newbest:
      generate_withdraw(oldbest)


      generate_announce(newbest)
      route_data_using(newbest)
```

(a) Regular Router Pseudocode

```
while session exists:
   (m, (p, n, r)) = get_message()
   oldbest = m_best(p, RIB)
   if m == ''ANNOUNCE'':
      while no memory available:
         evict_route(RIB)
      RIB = RIB + (p, n, r)
   if m == ''WITHDRAW'':
      RIB = RIB - (p, n, r)
   newbest = m_best(p, RIB)
   if oldbest != newbest:
      generate_withdraw(oldbest)
      if is_evicted(newbest):
         refresh_route(newbest)
      generate_announce(newbest)
      route_data_using(newbest)
```

(b) Forgetful Router Pseudocode

Figure 2: Pseudocode describing how a regular router and how a forgetful router would operate. These two code pieces are space-aligned to highlight the differences.

each R_p, where R_p is the subset of RIB entries that route to prefix p. When picking a "best" route for each prefix, BGP relies on a combination of custom routing policies and the BGP Decision Process [13], a universally agreed-upon ranking function. An example of a routing policy may be, "prefer routes learned from neighbor p over neighbor q." The BGP Decision Process is a multi-step procedure over the route attributes: *e.g.*, prefer routes with fewer AS hops to ones that have more; if a tie exists, prefer routes that were learned earlier than ones learned later; if another tie exists, prefer routes learned from ASes with a lower ID number.

In general, it is possible to think of these routing policies and the BGP decision process as being represented by a single metric that compares all the routes. This metric uses all the route attributes, along with these rules and policies, to rank them in the same order as in BGP. We define $m_{\texttt{best}}$ to be the metric that picks the routes to use for forwarding traffic to prefix p:

$$\begin{aligned} m_{\texttt{best}} : R_p &\rightarrow (p, n, r) \\ m_{\texttt{best}}(R_p) &\in R_p \\ \forall e \in R_p,\, e &\preceq m_{\texttt{best}}(R_p) \end{aligned}$$

The first two equations state that $m_{\texttt{best}}$ operates over R_p and always picks a RIB entry from R_p; the last one describes how the "best" RIB entry has the highest rank among all entries.

When the session closes, the router withdraws all routes advertised by the neighbor, re-computes its new set of best routes, and sends updates as needed. As new routes are learned and old ones are forgotten, the set of best routes will change over time. Each time a best route changes, update messages are generated for appropriate neighbors as long as the BGP session continues. See Figure 2(a) for pseudocode describing how an ideal router acts.

2.2 Evicting and Refreshing Alternate Routes

Routers have fixed amounts of memory and can only store a limited number of RIB entries. However, the BGP specification does not define how the protocol should behave when a router runs out of memory. Furthermore, BGP is memory intensive. It is *prefix-based*, and although this is more efficient than enumerating IP addresses, there are many prefixes. Moreover, BGP is a *path-vector* protocol, forcing routers to store the entire AS path for each router. In addition, BGP is a *policy-based* protocol, with numerous route attributes that influence the selection and propagation of routes. Finally, BGP is an *incremental protocol*: upon receiving an update message, a router must compare the new information with all previously learned routes to select the new best route; information cannot be arbitrarily discarded, since it may be needed at a later time.

We thus propose the following behavior whenever a router experiences a dearth of free memory: pick a (p, n, r) that is currently an alternate route and *evict* extraneous information from r that is ignored by $m_{\texttt{best}}$. We define an evictor ev with the following properties:

$$\begin{aligned} ev : (p, n, r) &\rightarrow (p, n, r') \\ \texttt{sizeof } r' &\leq \texttt{sizeof } r \end{aligned}$$

These equations tell us that ev modifies the additional routing information to possibly consume less memory. Moreover, modifying our RIB such that one of the entries has evicted information does not change the BGP decision process. See Figure 3 for an example RIB entry and how it could be compressed.

The fact that BGP's routing data can be compressed without affecting its decision process is non-intuitive. For example, although the decision process favors routes with fewer hops than those with more hops, BGP stores the entire AS path instead of the length. BGP must do this so when it advertises a route, loop detection can occur. Thus, while routes are not announced, they can be safely compressed; when they are in use, all their original routing information must be retrieved and sent to all neighbors.

Because the modified route contains enough information to allow $m_{\texttt{best}}$ to rank it, the routes chosen by this scheme always match the routes chosen by an equivalent, regular router. However, if a modified route suddenly becomes "best,"

it cannot be announced until the complete non-metric routing information is retrieved, since it might be relevant to other routers. Retrieving this route will require a *refresh*. Since routers only advertise their best routes to their neighbors, every alternate route is always some neighbor's best route, so the additional routing information will always be retrievable.

In order to decide which alternate routes to evict, we define an eviction mechanism $m_{\texttt{evict}}$:

$$\begin{aligned} m_{\texttt{evict}} : R &\rightarrow (p, n, r) \\ m_{\texttt{evict}}(R) &\in R \\ \forall p, m_{\texttt{evict}}(R) &\neq m_{\texttt{best}}(p, R) \end{aligned}$$

These equations state that the eviction mechanism can choose any route in the RIB, as long as it is an alternate and as long as it has not already had some of its routing information evicted. Pseudocode used to describe a router's behavior with this extension is provided in Figure 2(b).

It is important to note that *ev* does not have to keep the metric relevant information. For example, *ev* could simply delete all additional routing information. This would force the router to issue a set of refreshes when it could not identify the new best route. In fact, there is an interesting trade-off between ambiguity introduced to the metric and potential savings. While we do not explore this trade-off, it holds promise for future research.

2.3 Properties of Forgetful Routing

Forgetful routing has three important properties: it does not alter the BGP decision process, it is incrementally deployable, and it does not adversely affect convergence delay.

Because compressed RIB entries keep enough information to allow ranking, and because evicted information is always retrievable, a forgetful router acts exactly like a regular router after convergence. Although a network of forgetful routers will act differently than a network of regular routers during route exploration and route changes (due to the fact that different routes may be announced at different times, due to delays introduced by refreshes), the two systems will always converge to the same steady state solution.

Moreover, it is possible for an autonomous system to deploy forgetful routing and obtain significant memory savings even if no other AS uses it. Since BGP already has a route-refresh option [10], and since forgetful routing's only assumption about its neighbors is that they support a form of refreshing a route, it is incrementally deployable. It is important to note that (currently) the route-refresh message triggers all prefixes to be re-advertised. This introduces a considerable amount of overhead for a single prefix needing a refresh. However, a cooperative route filtering capability [14] is in development would allow individual prefixes to be refreshed.

In addition, forgetful routing does not significantly affect the propagation delay of routes; in fact, the total increase in convergence delay over an entire network is only the time of a *single* refresh. This property is best illustrated through the following example. Imagine a network with three routers, R1, R2, and R3, that can all route to prefix P. R1 depends on R2 to reach P, R2 depends on R3, and R3 has some mechanism to reach P; in other words, there is a chain of dependency. Now imagine that all alternates are discarded and R3 loses its primary route. While R3 is refreshing, it simultaneously sends a withdraw message to R2, allowing

```
PREFIX:        4.21.252.0/23
FROM:          194.85.4.55  AS3277
TIME:          2004-12-31 20:07:56
TYPE:          MSG_TABLE_DUMP/AFI_IP
VIEW:          0  SEQUENCE: 440
STATUS:        1
ORIGINATED:    Fri Dec 31 06:26:51 2004
AS_PATH:       3277 13062 20764 701
               6389 8063 19198
NEXT_HOP:      194.85.4.55
COMMUNITIES:   3277:13062 3277:65301
               3277:65307 20764:3000
               20764:3011 20764:3020
               20764:3022
```

Figure 3: A BGP RIB entry from a RouteViews table dump. All values other than "prefix" and "from" are represented by r in our abstract model. Note that while the path attributes would take at least 50 bytes to encode, the metric-relevant attributes listed here (time and AS path length) could be encoded in about 5 bytes.

R2 to start issuing its own refreshes *in parallel*. Likewise, R2 performs the same actions, allowing R1 to start issuing its own refreshes as well. By the time R3 has recomputed its best route and sent it out, R2 will have finished performing its refreshes and can propagate the new routing information instantly. The same holds true for R1. The net effect is that the additional end-to-end delay only increases by the time of resolving a single refresh. Given that typical convergence delay on the Internet is on the order of minutes, and given that a refresh may be processed in milliseconds, we feel that the additional delay is negligible.

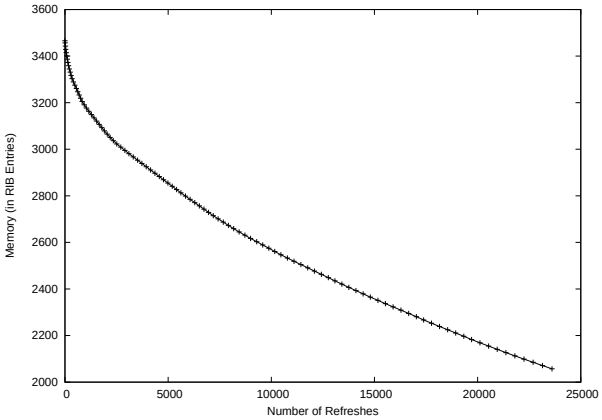
3. OPTIMAL OFFLINE ALGORITHM

Choosing an eviction policy for forgetful routing is a specialized instance of the cache-replacement problem. Since an eviction policy will depend highly on the RIB entries and update messages seen, it is important to evaluate different policies and their performance. In fact, some baseline is necessary even before evaluation of policies can begin.

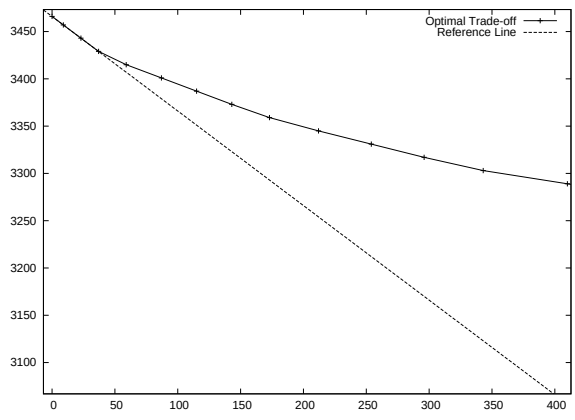
In order to investigate the trade-off curve and to obtain a gold standard for comparing other routing policies, we developed an offline algorithm that, given a RIB dump (*i.e.*, a set of (p, n, r) in the RIB at a specific time) and a stream of update messages, determines the best possible trade-off between available memory and frequency of refreshes. For simplicity, we assume that every eviction frees an entire RIB entry, and that all RIB entries consume the same amount of memory. The algorithm is offline because in order to calculate the theoretical best trade-off, it must have foresight of the future to influence decisions in the past.

3.1 Optimal Eviction Policy

The first step of the optimal algorithm computes the set usage times of all routes. That is, each route is annotated with the set of times that it will be selected as "best." This is calculated by performing a pass through an initial RIB dump and a stream of update messages, maintaining a hash table that maps from RIB entries to lists, where each RIB



(a) Optimal tradeoff from 1/1/2005 to 7/1/2005



(b) Close up with reference

Figure 4: The optimal tradeoff curve, with number of refreshes issued as a function of memory. The dotted line in Figure 4(b) has a slope of -1 and is meant as reference.

entry has its associated list updated whenever it is chosen as best. A simulated router is used for determining usage for each RIB entry e: A simulated router is used for determining usage for each RIB entry e:

$$\texttt{use}(e) = \{t_1, t_2, \ldots, t_n\}$$

Element t_i represents the i^{th} time that $m_{\texttt{best}}$ chose the RIB entry e.

Once this calculation is complete, the eviction policy is straightforward: always choose the alternate route that will be used last, or furthest in the future. Given that the current time is t when an eviction is about to occur, the route that will be needed furthest in the future can be calculated and $m_{\texttt{evict}}$ can be implemented via the following statements:

$$\texttt{nextuse}(e,t) = \begin{cases} 0 & \texttt{if } e \texttt{ is best} \\ \min_{t_i > t} \texttt{use}(e) & \texttt{if } \exists t_i > t \\ \infty & \texttt{otherwise} \end{cases}$$

$$m_{\texttt{evict}}(R) = \texttt{maxarg}_{e \in R}[\texttt{nextuse}(e,t)]$$

The function *nextuse* returns the earliest time after t when the route will be used. Note that 0 is returned if e is currently best (ensuring the algorithm picks an alternate route), and that routes that are never needed are given a time of infinity, making them the best eviction candidates.

A proof of optimality has been omitted due to length constraints, but the intuition behind the proof can be described concisely. By always choosing a route that is needed furthest in the future, we can guarantee that we cannot do any worse than if we picked any other route. This is because any route that is needed in the future will cause an eventual refresh and, in the interim time, lowers the total amount of memory needed. Given that routes are independent of each other in terms of their usage, it is not possible to pick a route that will be needed *earlier*, and yet somehow causes other routes to refresh *less* often; the route that is needed furthest in the future provides the longest interval of memory relief, allowing other routes to remain in memory longer before eviction.

Implementing this algorithm naively is easy: perform a pass through the RIB dump and update stream to calculate a list of usage times for every route, and use this information to evict routes. Implementing this scheme efficiently, however, is much more complex. Without going into details, lazy evaluation of evictions can be used to achieve the same results without as much computation; that is, instead of calculating which route needs to be evicted at the very moment when memory capacity has been reached, the simulation can note that a route was forgotten and later deduce the correct one by monitoring usage of alternate routes.

3.2 Evaluation on BGP Message Traces

The data set we use was obtained from RouteViews [15] and consisted of a BGP table dump on January 1st, 2005, and all BGP update information for six consecutive months. RouteViews was chosen primarily because it is publicly available, while ISP feeds are proprietary and difficult to obtain. There were approximately 270,000 different prefixes announced over that period. We randomly sampled 1% of them and their associated updates for our analysis. Our sampling is due to the fact that these simulations take significant computational time—on the order of years when making multiple scans through six month's worth of update messages.

It is important to note that the data obtained from RouteViews is not typical. RouteViews connects to many more neighbors than most routers and thus has many more alternate routes, upwards of 40 per prefix. Rather than filter the RouteViews data to represent a "typical" AS, we included all feeds to capture the dynamics of forgetful routing, as well as quantify how much memory savings are possible. Thus, although our estimates of the amount of memory saved may be optimistic, RouteViews allows us to examine the general impact of forgetful routing. We leave analysis of the gains typical ASes would see to Section 5.

Our results show that many alternate routes are never needed and can be safely discarded without causing refreshes. In the best possible case, an average of one alternate route per prefix is sufficient and will not cause any refreshes. The algorithm's trade-off curve can be seen in Figure 4(a). The first important item to note is the reduction in BGP table size. At zero refreshes, the amount of memory needed never exceeds 3,500 RIB entries, whereas a full table would require approximately 68,000 entries. In other words, if a router had foresight and sufficient additional computing power, it could

reduce its memory footprint size by 95%.

The second important item to note is the initial one-to-one trade-off between memory size and refreshes, as seen in Figure 4(b). Initially, lowering the total amount of memory by one unit causes one refresh. In this part of the curve, the alternate routes that are evicted in these cases belong to very stable prefixes, where alternates are rarely used. After total memory has decreased by about 200 units, freeing one unit of memory results in multiple refreshes. This occurs when we have evicted all the alternate routes for stable prefixes and we must start evicting routes for unstable ones. Since these alternate routes will be refreshed into memory much sooner, more refreshes will be needed in the same period of time.

Even at a memory size of 2048 memory units, the number of required refreshes is surprisingly small, given that sixteen million RIB entries were added and removed for this six-month interval through update messages. Furthermore, this memory footprint is 3% of the original size. All of these items demonstrate several important points: (i) that tremendous memory savings are possible without significant increases in bandwidth, (ii) that the majority of alternate routes are not used, and (iii) that the best way to minimize refreshes is to keep the alternate routes for unstable prefixes "in the cache."

4. ONLINE ALGORITHMS

Although the results of the offline analysis are promising, the algorithm is not feasible in practice; foresight of the future and infinite computing power are not luxuries afforded to today's routers. In this section, we devise efficient online algorithms that approximate the optimal results. We evaluate these online algorithms in two parts. First, we examine all algorithms under the assumptions that RIB entries have uniform size, that each eviction frees one RIB entry of space, and that memory overhead from additional data structures is negligible; this yields raw performance information for direct comparison with the optimal offline algorithm. Then, we re-examine the algorithms under realistic assumptions.

4.1 Least Recently Refreshed and Updated

When first devising an online algorithm, it is important to see whether a very simple scheme can achieve most of the savings. As such, our first strategy was constant-size allocation, where each prefix is given a fixed amount of memory in advance. By varying the amount of dedicated memory per prefix, and evicting the lowest-ranked routes for each prefix when memory is tight, the memory-bandwidth trade-off can be explored. We implemented such a scheme and found it to be too simplistic; prefixes with very stable primary routes are allocated too much space, while unstable prefixes are allocated too little space, leading to frequent refreshes.

Our next strategy was to evaluate the canonical algorithm for cache-replacement problems: Least Recently Used (LRU). Our LRU algorithm uses a doubly-linked list, where each cell has a RIB entry. RIB entries are added to the back (and removed when their associated withdraw is received), and are picked for eviction from the front. Thus, assuming four bytes per pointer with two pointers per linked list cell, all decisions can be made in constant time with $8n_r$ bytes of memory overhead, where n_r is the number of RIB entries. Unfortunately, $8n_r$ overhead is rather steep, especially considering our goal is to *save* memory, not consume it!

In order to achieve most of the benefits of LRU without the memory overhead, some approximation is needed. We thus devised a variant called Least Recently Refreshed (LRR). Under LRR, we always evict the least desirable route of the prefix that has not needed a refresh in the longest amount of time. In doing so, we can replace our doubly-linked list of *RIB entries* with a doubly-linked list of *prefixes*. Prefixes are put on the back of the list when they have just been discovered by the router and when they are just refreshed. This enables all operations to complete in constant time, while reducing memory overhead to $8n_p$, where n_p is the number of prefixes.

In addition, a variant of LRR called *Least Recently Updated* (LRUp) was implemented. This variant uses the time since the last update message as the ranking metric, rather than time since last refresh. It uses all the same memory structures as the LRR algorithm and the same computational operations.

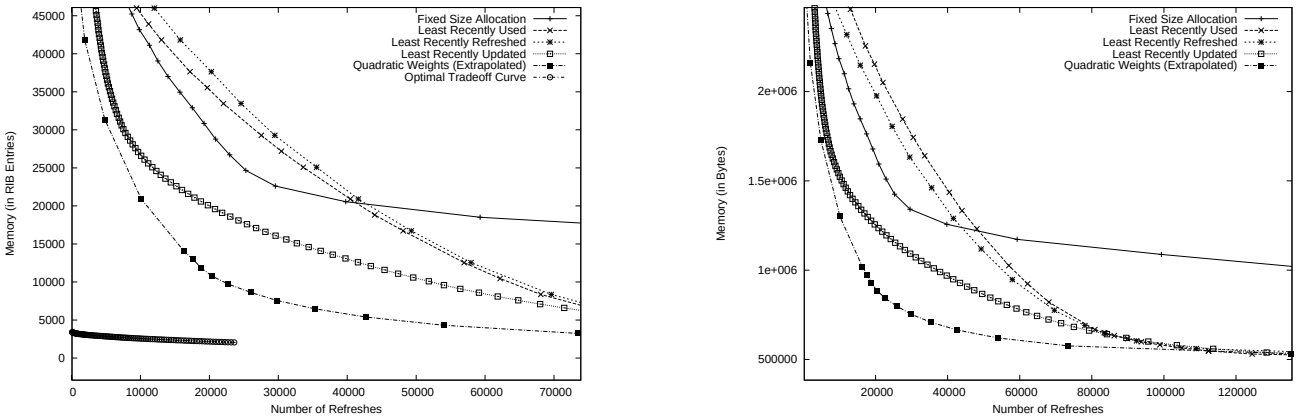
The results of fixed-size allocation, LRU, and LRR can be seen in Figure 5(a). Although fixed size allocation performs reasonably well when a large amount of memory is available, it quickly begins to fail as memory is driven lower and lower. From the figure, it is clear that simple schemes cannot capture the dynamics of routes and their refresh needs. Both LRU and LRR, on the other hand, perform competitively, within an order of magnitude of the theoretical best. Although LRR's data structures require much less memory, the performance is nearly identical to LRU. For a small sacrifice in the number of refreshes, a significant reduction in overhead can be achieved.

Surprisingly, LRUp performs much better than LRU or LRR. The reason behind this phenomenon lies in the fact that any update activity is usually a sign of route instability. By exploiting this fact, LRUp can anticipate which prefixes will need more alternate routes available in memory. Still, a sizable gap remains between the LRUp scheme and the optimal offline algorithm. Although some gap is inevitable, because the offline algorithm has foresight of the future, we wanted to explore whether a more sophisticated eviction policy could improve the effectiveness of forgetful routing.

4.2 Quadratic Weights

Although LRU, LRR, and LRUp perform well, they rely on just a single variable to rank routes for eviction. Moreover, none of these schemes consider the *number* of alternate routes available. Often, when a relatively stable prefix has a routing change, the router switches from the primary route to the first or second alternate route [16]. Keeping one or two alternate routes in the RIB, even for the relatively stable prefixes, can help reduce the number of refreshes. At the expense of additional computational complexity, an online algorithm can account for both of these factors: the time since the last refresh and the number of alternate routes.

In doing so, we devise another online algorithm which we call quadratic weights (QW). Instead of using time to order routes, we calculate a "goodness" value for each prefix p, according to the formula $n * (n-1) * t$, where n is the number of routes to p and t is the time since the last refresh. Note that other multiplicative factors could be used, such as linear or cubic. Depending on the dynamics of the network, such factors may understress or overstress the importance



(a) RIB Entry Comparison

(b) Memory Usage Comparison

Figure 5: The tradeoff between refreshes and memory capacity for various online algorithms. 2684 prefixes were used for these calculations (randomly sampled out of the 268480 total prefixes seen over a six month period).

of alternate routes.[4] After a factor is chosen, routes are ordered by goodness and then appropriately evicted.

Although the quadratic-weights algorithm only consumes $8n_p$ bytes of memory in additional data structures, it has $O(n_p)$ computational overhead. Since each prefix's t value is constantly changing (even if no corresponding update messages have been received), all prefixes must have their goodness values re-computed during an eviction.

The results of this algorithm can be seen in figure 5(a). Due to the computational overhead, the quadratic weights algorithm was calculated over a 0.1% sampling and then extrapolated for our 1% sampling. We believe this extrapolation to be fairly accurate, as extrapolating the LRU algorithm from the same 0.1% sampling back to a 1% sampling resulted in an almost perfect fit. Quadratic Weights is better than any of the online algorithms, but not considerably; in particular, it is somewhat close to the LRUp algorithm. Given that LRUp is very simple to implement in practice, it is unclear that if a QW approximation algorithm would be desirable, as it would most likely be more complicated.

4.3 Memory Usage

In order to understand how these algorithms would operate in practice, we extrapolated total memory usage for each online algorithm. Drawing on analysis of memory usage on several commercial routers, we assumed an average size of 45 bytes per RIB entry. Any memory overhead from the algorithms' data structures was taken into account. Additionally, one byte per RIB entry of overhead was added for booking which routes were forgotten. Lastly, we assumed that evicting a route leaves behind four bytes of metric-relevant data.[5].

The results can be seen in Figure 5(b). For reference, a regular router would need to consume approximately 353 megabytes of space to store all of the routes, or 3.53 megabytes at a 1% sampling.

It is interesting to note that all the algorithms hardly deviate from their relative positions. Moreover, the memory savings are still quite substantial. Approximately 10% of the memory savings is lost due to overhead costs, most of which stem from the one additional byte per RIB entry and the four bytes of metric-relevant data left behind. If one bit was used instead of one byte (to mark whether the routing data was represented in a compact format or not) and fewer bytes of metric-relevant data were used, this gap could be closed even further. Exploring the trade-off of throwing out some metric data (and thus making it possible to have a set of possible next best routes) and the increase in refreshes that would be required is an area of potential future work.

5. EXPECTED MEMORY SAVINGS

Given that forgetful routing's ability to save memory relies on network topology, it is important to investigate how much savings real routers might see. Moreover, observing how these gains vary by AS provides insight into the types of systems that would see the most benefit from our scheme.

5.1 Challenges for Realistic Evaluation

Ideally, we would like to obtain the BGP message streams sent to a router and use them to quantify the dynamics of forgetful routing. By sampling these feeds for routers from ISPs, business customers, university networks, *etc.*, we could build an extremely accurate picture of how forgetful routing would affect a wide range of BGP speakers, both in terms of memory and in terms of refreshes. Unfortunately, it is very difficult to obtain accurate information about real-world memory savings. Knowing how ASes' connect and what messages they receive reveals valuable business information. For this reason, companies do not publish their feeds nor do they make them easily accessible.

While we do have RouteViews as a source, it is not representative of any typical AS. This is because it connects to over forty neighbors and receives full routes (i.e., routes for

[4]Our choice of a multiplicative, quadratic factor of `n(n-1)` came from theoretical analysis that made simplifying assumptions about the types of updates seen and their frequency.

[5]Of these four bytes, one can be allocated for AS path length, one for ranking the time the route was learned against other routes for this prefix, and two for compactly representing the other BGP metric variables

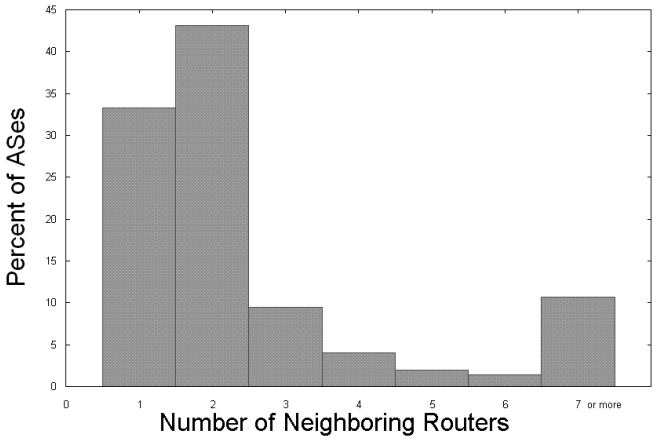


Figure 6: Percent of ASes with a given number of connections.

all destination prefixes) from many of them. While we can use RouteViews to evaluate different algorithms against each other, we cannot use it to evaluate expected gains. Since obtaining dynamic feeds is out of the question, we cannot deduce real world estimates of the refresh rates of forgetful routing. However, static dumps of various RIBs are available, allowing us to quantify memory savings. First, we analyzed the RIB of the border router that connects Princeton University to the rest of the Internet. The Princeton campus network connects to four other networks. Based on a dump from April 6th, 2006, we observed that, on average, the Princeton network has 2.4 alternate routes per prefix. Thus a memory savings of nearly 70% would be achievable for this router.

Next, we looked at BGP data from looking glass servers. A looking glass server is a router that additionally runs a publicly available interface (such as a website or a telnet connection) permitting anyone to query routing information from it. The results of analyzing five looking glass servers [17] can be found in Table 1. For each of these ASes except AS 7018, twenty prefixes were randomly sampled and the average number of RIB entries calculated. For AS 7018, the first 20,000 prefixes in its RIB table were used to compute its average. All of these ASes represent domains with high connectivity. Unfortunately, looking glass servers are not very accurate indicators of what real routers' RIB tables look like. Because BGP information has business value, as describes how different competing organizations form alliances with each other, looking glass servers tend to omit many RIB entries. For example, AS7018, which is AT&T, is reported to have an average of 1.2 available routes per prefix, a number which is absurdly low for a tier-1 provider.

5.2 Inference of BGP RIB Sizes

Looking at such a small number of static dumps is very limiting. Without a wider breadth of routers' RIBs, it is difficult to quantify what typical gains an AS might see. However, we can infer other routers' RIBs without direct access to them. If we know the Internet's topology, and if we know where prefixes originate, and if we know the routing policies used between ASes, we can simulate the propagation of routing information across all the unknown ASes' routers and calculate what their RIBs look like.

In order to infer Internet topology, all that is needed is a RIB dump from any well-connected router. By looking at all AS paths for all routes in the dump, one can deduce that there exists a link between each adjacent pair of ASes (*e.g.*, if a router receives a route advertisement that has an AS path of (41, 65, 45), then 41 and 65 connect to each other, as do 65 and 45). Doing so results in a conservative count—there may be some links that were not represented in AS paths, depending on the reference router's isolation from other networks. Thus, any Internet topology inference based on this scheme will, if anything, undercount various ASes' RIBs in a simulation, as well as the number of alternate routes available to prefixes.

Calculating where prefixes originate is a rather simple task. For each RIB entry in a router dump, observe the prefix that it routes to as well as the last AS in the AS path. By definition of BGP's behavior, this AS will be the originator. Problems that may arise include undercounting the number of prefixes available. Since these prefixes and their origins are derived from a RIB dump, a router that is partitioned off from a significant section of the Internet may not know where certain prefixes originate, or may not know that they even exist. Likewise with Internet topology, any inference based on this scheme will undercount the number of RIB entries in a simulation, but will not affect the count of alternate routes available to known prefixes.

Inferring an AS's routing policies is difficult. By observing sets of AS paths, one can examine how they constrain the possible relationships between organizations and attempt to model them. A scheme that infers extremely liberal routing policies will overcount the number of alternate routes per prefix, while a scheme that infers extremely conservative routing policies will undercount the number of alternates.

To derive routing policies, we used the Gao inference algorithm [18] on a BGP RIB dump obtained from RouteViews [15] on February 10th, 2005. The Gao algorithm defines three different types of business relationships: provider-customer, peer-peer, and sibling-sibling. Provider-customer relationships represent one AS purchasing connectivity from another AS. Providers almost always advertise all their routing information to their customers, as they typically charge their customers for traffic that flows between them. Peer-peer relationships represent two competing organizations that connect to each other in order to achieve greater reachability. Peers generally only advertise their customer-learned routes and do not re-advertise routes they learn from other peers. By doing so, their customers have great reachability (and thus more traffic, generating more revenue), and they never need to transit traffic meant for another peer (which would consume network resources without generating revenue). Sibling-sibling relationships indicate that the two ASes are actually controlled by the same organization, either through mergers or buyouts. In some sense, siblings should be treated as one large AS.

The results from our analysis can be seen in Figure 6. From this figure we can identify three different categories of autonomous systems:

- Single-homed ASes are autonomous systems that purchase their Internet connectivity from a single, upstream provider. Based on our data, about 30% of all ASes fall into this category. Because they only have one provider, they never have alternate routes. For

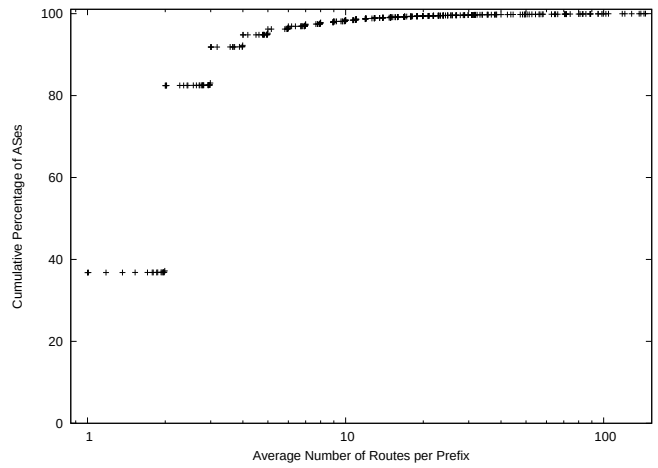


Figure 7: Percent of ASes with a given average number of routes per prefix.

these systems, forgetful routing offers no benefits.

- Multi-homed ASes are autonomous systems that purchase their Internet connectivity from two or more upstream providers. Approximately 40% of all autonomous systems belong to this labeling, including the Princeton network mentioned earlier. The upstream providers are used either as an emergency backups or for load balancing. Multi-homed ASes are thus candidates for forgetful routing.

- Providers are autonomous systems that have excellent connectivity and typically sell their reachability to single-homed or multi-homed customers. Some of these providers may themselves be customers of larger ISPs. Approximately 30% of all ASes fall into this category. For these systems, forgetful routing could offer an order of magnitude of memory saved, or more.

Additionally, by analyzing the business relationships between ASes, we inferred the number of alternate routes available to each autonomous system for each prefix. Using the business relationship data obtained from the Gao algorithm, along with the BGP RIB data from RouteViews to determine prefix origins, we simulated BGP announcements over the derived topology. We assumed that providers always advertise information to their customers, that customers do not advertise information to their providers (unless they are originating a prefix), that peers only advertise their customer learned routes, and that sibling ASes advertise all their information to each other. Figure 7 shows the results. This CDF plot has three key areas:

- *The No-Savings Zone.* The first area contains all the single-homed ASes mentioned earlier, which never have any alternate routes. This explains the large vertical offset of 30% in the graph. We also see that there are a few autonomous systems that connect to multiple ASes, but have virtually no alternate routes to choose from.

- *The Savings Zone.* The second area consists of the large spike from 2 to 20 on the x-axis. The immediate jump at 2 represents multi-homed ASes, which have alternate routing information for backup connectivity. Onward until 20 we see a spectrum of values, representing other multi-homed ASes and providers.

- *The Overestimated Zone.* After 20 we see that there are a small number of ASes which appear to have extreme numbers of alternate routes. This part of the graph, from 20 onward on the x-axis, is actually an artifact of our simplistic assumptions about routing policies. Many of these ASes in this region of the graph connect to peering points, where hundreds of other organizations also connect. Using the RIPE [19] Whois database, we discovered that many of these peers have extremely strict import and export policies to avoid the RIB table explosion as seen in the graph.

AS	Looking Glass
AS 3333	9.0
AS 6395	1.7
AS 6461	5.7
AS 7018	1.2
AS 9607	4.0

Table 1: A comparison of the number of RIB entries per prefix.

5.3 Discussion of Potential Savings

While the gains can vary from as low as nothing to as high as an order of magnitude, it is important to evaluate these gains in the context of the likelihood of deploying forgetful routing. *Single-homed stub ASes* only connect to one upstream provider and receive all routes from their provider; since they have no alternates, they derive no benefit from forgetful routing and would thus not deploy it. *Multi-homed stub ASes*, like Princeton, generally have low connectivity, but receive almost all routing information from every connection. Thus, these ASes may see a reduction in RIB size from 33% to 75%. These ASes are likely to have older routers with smaller amounts of fixed memory. Moreover, since their ISPs would most likely have the route-refresh capability, they could deploy forgetful routing and obtain these memory savings.

It is also important to note that, for stub networks, forgetful routing has no side-effects. Performance is unchanged, as well as convergence delay, as stubs never re-transmit routing information. For these networks, forgetful routing can truly be transparently deployed.

Transit providers generally have high connectivity, but do not always receive all routing information from every connection. Our analysis estimates that providers could see a reduction in RIB size of 75% to 95%. Moreover, if forgetful routing is deployed, incentives will then exist for increasing connectivity further, allowing for greater reliability without significant increases to memory usage. In fact, many of those ASes who peer with 100+ neighbors, but currently use extremely strict filters to avoid RIB table explosion, could use forgetful routing to regain the lost connectivity at a mere fraction of the memory cost.

It is important to note that providers may see additional gains not presented in this analysis if they run forgetful routing *inside* the AS. Many ASes use a popular variant of BGP

called *IBGP* to internally store routes learned from their neighbors. Under IBGP, every router within a single AS has a BGP session with every other router and all information is shared between all routers, providing a global view of the network. However, IBGP's memory requirements grow very quickly; for n routers in a network, each one would need up to n times more memory. Several solutions have been proposed to scale back on the burdens IBGP introduces, with the most notable being Route Reflectors and AS Confederations [20, 21]. Route reflectors operate by dividing routers into two subsets: route reflectors and route reflector clients. Clients connect to reflectors and rely on them to re-advertise their routes, as well as to learn new routes [11]. However, reflectors make networks more vulnerable to point failures and can cause a inconsistent routing decisions, persistent loops, and route oscillations [22, 23]. AS Confederations partition an AS into sub-ASes, where each sub-AS maintains a full mesh within itself, with BGP sessions the between border routers of each sub-AS. However, not only do AS Confederations make networks less robust, they can also lead to instances of sub-optimal routing [23]. With forgetful routing, it could be possible to maintain full IBGP connectivity without inflating the memory requirements.

The most important insight from our analysis can be summed up in one statement: having n neighbors does not always imply a factor of n savings. The type of network, such as multi-homed stub, provider, *etc.*, greatly affects the total amount of savings. However, in most cases, our results indicate that significant savings are possible. It is difficult to say whether our estimates for memory savings are high or low. On the one hand, our topology is almost certainly incomplete, undercounting total savings. On the other hand, our inferred routing policies are liberal, overcounting total savings. Because of the difficult nature of inference, covert policies and routing data, *etc.*, this area is one of potential future work.

6. RELATED WORK

CRIO [24] is a mechanism which uses IP tunneling to reduce the memory needed by BGP significantly, at the expense of longer paths. It operates by having tier-1 ISPs announce "virtual prefixes" that are very large (*e.g.*, /8's), and using tunnels to directly connect routers as they forward packets. However, CRIO requires a globally deployed system to operate and changes the way BGP functions. Our method is incrementally deployable on a per-router basis and can be completely implemented using the BGP route-refresh option. Additionally, CRIO sometimes reduces path diversity and increases path lengths, while our mechanism does not reduce the resilience and natural efficiency of BGP.

A similar endeavor is the atomized routing project by Caida [25]. Atomized routing groups prefixes together into units called 'atoms' if they share the same AS path in all their routes through any router in a network. These atoms are globally computed and used to route packets. Studies have found that up to 78% of memory can be saved using this technique. However, atomized routing has the same issues as tunneled-BGP—it is not incrementally deployable and it requires modification to BGP.

Draves et al. worked on a similar problem of reducing the Forwarding Information Base (FIB) size of routers [26]. The FIB stores forwarding information for each prefix and is usually installed on fast line cards, unlike the main memory used by the RIB. Draves et al. proposed an algorithm that constructs optimal trees to store this information, guaranteeing maximal FIB savings. This work, however, is vastly different from ours. It focuses on an entirely different memory system where all information is local and operates using prefix aggregation. Unlike the FIB, the RIB's information may need to propagate to others and thus cannot be aggregated.

Research on shifting the burden of memory and computation to centralized servers has been proposed by Caesar et al. in the development of a Routing Control Platform (RCP) [27]. The RCP acts as a central server that collects BGP data from all neighboring routes in the same AS. The RCP then has access to global information and can make global routing decisions. It sends forwarding information back to each router, moving the RIB memory burden and computation to a single machine. While this can allow for different compression mechanisms (such as storing all unique AS path tags across all routers in a single table, eliminating duplicates), such schemes have not been fully developed yet. It is also important to note that a system such as an RCP may run into memory problems itself if it attempts to store all routes for an entire AS. Thus, a system such as the RCP may benefit from schemes like forgetful routing that are modified for RCP to RCP communication.

Finally, earlier work on the EIGRP protocol [28] is related to Forgetful Routing. EIGRP uses distance-vector routing along with a query/reply system to ensure loop-free routing. For every route, each router stores each neighbor's metric associated with that route. The "best" route chosen is the one with the lowest metric. Whenever that route becomes unavailable, the router checks if an alternate exists with the same metric, and if so, uses it. Otherwise, it propagates a query message to all its neighbors and recomputes its best route after it has heard back from all of them. Although EIGRP has a similar "refresh" mechanism like forgetful routing, it has two distinct differences. First, since EIGRP is a distance-vector protocol, it suffers from the "count to infinity" problem that must be resolved by setting a maximum hop count. Second, unlike forgetful routing's refresh mechanism, EIGRP's refreshes are executed *in series* rather than *in parallel*, resulting in convergence delays that increase linearly with network size.

7. CONCLUSION

Today's routers are susceptible to a wide range of memory problems. When a routing table overflows, the router may enter unspecified behavior, including freezing, rejecting new routes, or entering into an infinite loop. It is not always possible to upgrade router memory, due either to costs or limited access to the machines. Furthermore, best common practices used by network operators only partially protect routers from memory overflow, as they must often sacrifice safety for operability.

Our proposal of forgetful routing offers a unique solution to the problem without modification to BGP. As memory is needed, alternate routes are forgotten and requested back as needed. Since best routes are never forgotten, and an alternate route is some neighbor's best route, the route refresh option in BGP can be used to implement this mechanism. An offline analysis of BGP data shows that there is much memory that can be saved; most alternate routes are never used. This analysis motivated the development of an online

algorithm that runs in constant time for each message and approximates the optimal solution well. We believe that our scheme can significantly improve the scalability and robustness of IP routers in the future.

Acknowledgments

The authors would like to thank Nick Feamster, Anja Feldmann, Randy Bush, Changhoon Kim, and the anonymous reviewers for their feedback and comments. This work was supported by HSARPA grant 1756303, and a URP grant from Cisco.

8. REFERENCES

[1] D.-F. Chang, R. Govindan, and J. Heidemann, "An empirical study of router response to large BGP routing table load," in *Proc. Internet Measurement Workshop*, 2002.

[2] T. Bu, L. Gao, and D. Towsley, "IPv4 address allocation and the BGP routing table evolution," *ACM Computer Communication Review*, vol. 35, pp. 71–80, January 2005.

[3] G. Huston, "CIDR report." Web site, `http://www.cidr-report.org/`.

[4] V. Bono, "7007 explanation and apology." `http://www.merit.edu/mail.archives/nanog/1997-04/msg00444.html`.

[5] W. F. Slater III, "The Internet Outage and Attacks of October 2002." Chicago Chapter of the Internet Society, November 2002. `www.isoc-chicago.org/internetoutage.pdf`.

[6] A. Popescu, B. Premore, and T. Underwood, "The anatomy of a leak: AS9121," May 2005. NANOG presentation, `http://www.nanog.org/mtg-0505/pdf/underwood.pdf`.

[7] N. Feamster, "Interdomain routing correctness and stability." `nms.csail.mit.edu/6.829-f05/lectures/L5-rtg_correctness_slides.ppt`.

[8] R. Bush. Email conversation, April 2006.

[9] K. Hubbard, M. Kosters, D. Conrad, D. Karrenberg, and J. Postel, "Internet registry IP allocation guidelines," November 1996. RFC 2050.

[10] E. Chen, "Route refresh capability for BGP-4," September 2000. RRC 2918.

[11] J. W. Stewart, *BGP4 Inter-Domain Routing in the Internet*. Addison-Wesley, 1998.

[12] "How BGP Routers Use the Multi-Exit Discriminator for Best Path Selection." `http://www.cisco.com/warp/public/459/37.html`.

[13] Cisco, "BGP best path selection algorithm." `http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a0080094431.shtml`, August 2005.

[14] E. Chen, "Cooperative route filtering capability for BGP-4." `http://www.ietf.org/internet-drafts/draft-ietf-idr-route-filter-13.txt`. IETF Internet Draft, expires September 2006.

[15] U. of Oregon, "RouteViews Project." `http://www.routeviews.org`.

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

[17] NANOG, "Looking glass sites." `http://www.nanog.org/lookingglass.html`.

[18] L. Gao, "On inferring autonomous system relationships in the Internet," *IEEE/ACM Trans. Networking*, vol. 9, pp. 733–745, 2001.

[19] "RIPE network coordination centre." `http://www.ripe.net`.

[20] T. Bates, R. Chandra, and E. Chen, "BGP route reflection - an alternative to full mesh IBGP," April 2000. RFC 2796.

[21] P. Traina and D. McPherson, "Autonomous system confederations for BGP," February 2001. RFC 3065.

[22] A. Basu, C. Ong, F. Rasala, B. Shepherd, and G. Wilfong, "Route oscillations in IBGP with route reflection," in *Proc. ACM SIGCOMM*, August 2002.

[23] R. Dube, "A comparison of scaling techniques for BGP," in *ACM Computer Communication Review*, vol. 29, July 1999.

[24] X. Zhang, P. Francis, J. Wang, and K. Yoshida, "Scaling IP routing with the core router-integrated overlay," in *Proc. International Conference on Network Protocols*, November 2006.

[25] "Atoms - atomised routing." `http://www.caida.org/projects/routing/atoms/`.

[26] R. Draves, C. King, S. Venkatachary, and B. Zill, "Constructing optimal IP routing tables," in *Proc. IEEE INFOCOM*, March 1999.

[27] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, "Design and implementation of a routing control platform," in *Proc. USENIX/ACM Symposium on Networked Systems Design and Implementation*, June 2005.

[28] "Enhanced interior gateway routing protocol." `http://www.cisco.com/warp/public/103/eigrp-toc.html`.