Available online at www.sciencedirect.com

SCIENCE DIRECT®

Simulation Modelling Practice and Theory 12 (2004) 187–216

SIMULATION MODELLING PRACTICE AND THEORY

www.elsevier.com/locate/simpat

Evaluation of efficient security for BGP route announcements using parallel simulation

David M. Nicol [a,*], Sean W. Smith [b], Meiyuan Zhao [b]

[a] *Coordinated Science Laboratory, Department of ECE, Illinois at Urbana-Champaign, Urbana, IL 61801, USA*

[b] *Department of CS, Institute for Security Technology Studies, Dartmouth College, Hanover NH 03755, USA*

Received 22 April 2003; received in revised form 22 September 2003; accepted 17 October 2003
Available online 5 June 2004

Abstract

The Border Gateway Protocol (BGP) determines how Internet traffic is routed throughout the entire world; malicious behavior by one or more BGP speakers could create serious security issues. Since the protocol depends on a speaker honestly reporting path information sent by previous speakers and involves a large number of independent speakers, the Secure BGP (S-BGP) approach uses public-key cryptography to ensure that a malicious speaker cannot fabricate this information. However, such public-key cryptography is expensive: S-BGP requires a digital signature operation on each announcement sent to each peer, and a linear (in the length of the path) number of verifications on each receipt. We use simulation of AS models derived from the Internet to evaluate the impact that the processing costs of cryptography have on BGP convergence time. As the size of these models grows, inherent memory requirements grow beyond what is normally available in serial computers, motivating us to use distributed memory cluster computers, just to hold the model state. We find that under heavy load the convergence time using ordinary S-BGP is significantly larger than BGP. We examine the impact of highly aggressive caching and pre-computation optimizations for S-BGP, and find that convergence time is much closer to BGP. However, these optimizations may be unrealistic, and are certainly expensive of memory. We consequently use the structure of BGP processing to design optimizations that reduce cryptographic overhead by amortizing the cost of private-key signatures over many messages. We call this method Signature-Amortization (S-A). We find that S-A provides as good or better convergence times as the highly optimized S-BGP, but without the cost and complications of caching and pre-computation. These experiments—whose memory demands easily exceed 10Gb—are made possible using

[*] Corresponding author.
E-mail address: nicol@crhc.uiuc.edu (D.M. Nicol).

1569-190X/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.simpat.2003.10.003

parallel simulation. They show that it is is possible therefore to minimize the impact route validation has on convergence, by being careful with signatures, rather than consumptive of memory.

Keywords: S-BGP; Inter domain routing; Parallel simulation for routing

1. Introduction

The Internet is comprised of a large number of Autonomous Systems (AS) that establish global connectivity by cooperatively sharing traffic. Traffic originating in one AS may end up being carried by the internal networks of several different ASes enroute to its destination. The routing of traffic across ASes is established by cooperative execution of a distributed protocol called the Border Gateway Protocol (BGP) [33]. Gateway routers that connect ASes (called BGP speakers), execute the protocol. This paper concerns BGP, optimizations to known solutions for adding security to its route announcement mechanism, and the evaluation of these optimizations using parallel simulation.

Lack of security is a serious concern that has been recognized for some time, e.g. [27,28]. The root cause of the problem is that BGP speakers trust the messages they receive, and trust other purported BGP speakers to be reliably executing the BGP protocol according to specifications. A comprehensive analysis of the security vulnerabilities in BGP is developed by Murphy in [21], and we echo key points of that analysis. The nature of the BGP protocol is that the messages a speaker sends are extensions of messages that the speaker earlier received. A compromised speaker can insert false information into the messages it sends; as that information is accepted and propagated, network performance may suffer:

- Data may be delayed in its delivery, or even prohibited from being delivered.
- Views of network connectivity may be incorrect.
- Data may be falsely routed through a portion of the network designed to eavesdrop, or even modify the data.
- Fed by false information, the BGP protocol itself may begin to misbehave in such a way that stable routes to networks are not constructed.

Murphy points out that that BGP has three fundamental vulnerabilities. The first is that it fails to ensure the integrity, freshness, and source authenticity of messages between speakers, the second is that it fails to validate the authority of a speaker to even participate in the BGP protocol, and the third is that it fails to ensure the authenticity of the information conveyed in BGP messages. The first issue can be addressed by hardening peer-to-peer communication with IPsec [4]. The second and third issues are addressed by a protocol known as S-BGP [13].

S-BGP provides a comprehensive treatment of prefix ownership and route authentication, including identification of needed public-key infrastructure, and techniques for embedding S-BGP extensions within the existing BGP specification in order to support incremental deployment. The security it offers is limited to route authentication—a rogue cannot fabricate routing information and inject it into the network. It does not protect against insider attacks on a router's route selection policy, its filtering policies, or the timing of the dispersion of the route announcements it does make. Still, the main concerns about deploying S-BGP today are not so much what it does not do as they are the difficulty of implementing and maintaining the public-key infrastructure needed to support it. Nevertheless, the vulnerability of the routing infrastructure is a timely concern, and solutions are likely to involve digital signatures of route announcements. Against this backdrop, our contribution is to show that the cryptographic processing costs of route authentication can adversely affect BGP behavior and to develop a low-impact solution to that problem.

Prior analysis of S-BGP identified its resource requirements [13]. Our contribution is to study how its additional computational load affects *convergence*—the speed at which BGP finds and distributes good routes. Using a detailed simulation model, we find that for highly connected routers under high load—e.g. a rebooting Tier 1 speaker—use of S-BGP significantly lengthens convergence time. Longer convergence reflects increased instability, and can cause degraded network performance. Consequently we examine S-BGP's cryptography costs and identify steps where performance improvements might be effective. We describe a new optimization that reduces the number of expensive cryptographic operations; using simulation we observe that with our technique BGP's convergence is close to that when no cryptography is used at all. We also consider aggressive route caching optimizations suggested in and inspired by the S-BGP literature, and find that convergence is no better than using our technique (without caching). Thus we have identified a new way of securing BGP path announcements, without significantly affecting convergence time, and without demanding significantly more router memory for caching validated and/or signed routes.

The behavior of convergence time is very much affected by the topology and logical relationships between autonomous systems. To obtain an understanding of how cryptographic processing costs affect convergence in the actual Internet, we need to simulate BGP behavior on the largest models of systems possible. The memory demands of a detailed BGP simulation are large—every BGP router stores the last advertised route for every network prefix, from each of its peers. This implies that the memory needs of the simulation are proportional to the product of the number of routers, average length of BGP advertisement, advertised prefixes, and average router connectivity. The simulator we use for our study (SSFNet) [3] can use an ordinary desktop computer to handle a network with 100 ASes, 2 prefixes per AS. For systems much larger than this we use parallel simulation on a distributed memory cluster computer. It is instructive to analyze the effectiveness of parallel simulation on our problem. We find that we benefit more on our experiments from the simple fact of being able to do them (using the available memory), than we do from the additional processing power. The simulator we use for our study is written entirely in Java, and so is portable to any computers that run Java—this is true even for the distributed version, which uses native Java methods for inter-process communication.

The remainder of the paper is organized as follows. Section 2 gives a brief overview of the BGP protocol, then Section 3 describes S-BGP, an existing proposal for securing BGP route announcements. Section 4 reports on benchmarking of cryptographic overhead, and Section 5 uses simulation to look at how this overhead affects BGP convergence time. We describe how to amortize route announcement signature overheads in Section 6. In Section 7 we discuss the parallelization of our problem class under SSFNet, and we summarize our conclusions in Section 9.

2. BGP

We first quickly review critical aspects of the BGP protocol.

An AS manages subnetworks, each one described by an IP *prefix*—a fixed pattern of the n highest order bits shared by all devices in the subnetwork. This n may vary from prefix to prefix. Packet forwarding is based on prefixes: given a packet's IP address, a router searches its *forwarding tables* for the longest prefix that contains it, and forwards through the port the table associates with that prefix. Routers that connect ASes use BGP to construct and maintain their forwarding tables; these routers are called BGP *speakers*. A BGP speaker communicates with a set of other BGP speakers, known as its *peers*. A speaker sends an **Update** in order to announce a new preferred route to a prefix p. The route is described as a sequence of AS ids. For instance, AS B_0 uses $B_0, B_1, B_2, \ldots B_k$ to announce the route it prefers to reach a prefix owned by B_k.

One way **Update** messages occur is when an AS announces prefixes it *originates* (i.e, owns) to its peers. This typically occurs when the speaker reboots. Another way **Update** messages get generated is related to the change of connection state between two peers. Peers maintain logical *sessions* with each other; message traffic across a session (including **KeepAlive** messages mandated by the protocol) informs the endpoints of their partner's liveness. Sessions do go down, for a variety of reasons. When a session is established (or re-established) the endpoints share their entire routing tables with each other, in the form of a large number of **Update** messages. Processing of an **Update** for prefix p may itself generate a number of new **Updates**. This happens if the AS path reported is the basis for a more attractive path; the recipient selects the new path, appends its AS identity to it, and announces the extended path to one or more of its peers.

Update messages between peers are rate-limited with a parameter known as the Minimum Route Advertisement Interval (MRAI). The MRAI is formally defined to be the minimum time that must elapse between sending successive **Updates** of the same prefix. In practice it is implemented as the minimum amount of time which must elapse between successive bursts of messages from one peer to another (particular) one. A speaker allocates an MRAI timer for each of its peers. The default value used throughout the Internet is 30 s, which is also the value assumed throughout our experiments.

At configuration time a speaker is encoded with a policy that governs how it selects preferred routes. Shortest-number-of-hops is a commonly adopted policy. A speaker also uses a configuration-time policy to select the subset of its peers to receive an **Update**. That policy reflects business relationships between the ASes the peers represent.

When a speaker announces a route to prefix p it implicitly *withdraws* its preference for the last route it announced to the same prefix. The recipient, having seen the old route and the new, understands that the sender no longer supports the older route. An **Update** message may also simply declare the route to a prefix to be withdrawn, without specifying an alternative preferred route. A route withdrawal can cause its recipient to generate an **Update** message; this occurs if the withdrawn route was the basis for the recipient's preferred route to the named prefix. A speaker saves the last announcement from every peer, for every prefix; receiving a route withdrawal it may find at hand another path to the affected prefix, and will announce the best of these.

A detailed description of BGP and its operation can be found in [33].

As we have seen, a route (or withdrawal) announced by one speaker for prefix p can initiate a wave of announcements (or withdrawals) by other speakers about p. The hope and expectation is that the wave eventually dies out. The length of time required for the wave to die out entirely is call the *convergence time*.

Studies of BGP have considered questions of stability (whether convergence is ever reached) [5,15,26], the policies that govern route selection [7,16,32], and convergence [6,8,14,16,17,22,34]. Convergence is of interest because the quality of connections to a prefix are degraded during the transient period when those connections are changing. For this reason various optimizations have been considered to control and accelerate convergence. Our study considers how adding cryptographic operations to BGP affects convergence time, and identifies techniques to minimize that effect.

3. S-BGP

Considered in its full generality, the problem of securing BGP from both inside and outside attack is daunting. As a consequence of autonomy, no speaker should necessarily trust the word of any other speaker. As a consequence of distribution, no speaker can possess a complete, current view of the system. As a consequence of size, the system as a whole can exhibit behavior not predictable when analyzed on a small scale.

The main security issue is directly identified. Each speaker constructs its forwarding table via hearsay: in general, the only knowledge B_0 has of a particular path $B_0, B_1, B_2, \ldots, B_k$ to a prefix p is the fact that B_k claims to originate p, and that along the way, each B_i forwards to B_{i-1} the commitment of $B_{i+1} \ldots B_k$ to participate in this path. Consequently, nothing prevents a malicious speaker Q from simply fabricating a claim to originate a prefix, or from fabricating a route to a prefix and announcing that to its peers. If the fabrication is sufficiently attractive, this misinformation will propagate throughout the network, as a subpath in other paths.

The natural approach to stopping such forgery is to use *digital signatures*. A digitally signed message typically contains the message m and the private-key encipherment $s(h(m))$ of a cryptographic hash $h(m)$ of m. (Section 4 gives more details.) If a party A receives a tuple $(m, s(h(m)))$ allegedly from party B who allegedly possesses a certain key pair, then A can extract m, reconstruct $h(m)$, and use the public key to verify that $s(h(m))$ matches $h(m)$. If $h(m)$ matches the signature, then A can conclude that the party with the matching private key sent the message.

To conclude that this party was B, however, A needs to know that B really possess that public key. Typical PKI achieves this via a *certificate*, an electronic document that binds identity information with a key pair. Party A receiving the tuple $(m, s(h(m)))$ from party B validates the message using a certificate that gives B's identity and B's public key. If A trusts this certificate, then verification of the signature on m tells A that B sent this message and that the message was not tampered with.

A must also validate the certificate, itself a digital message. Typical PKI achieves this by having another authority vouchsafe for the certificate, through another digital signature of course, applied to B's certificate; typically, this chain of verification is carried higher and higher up a trust ladder up to the root of a trust tree. Fortunately most of this verification need only be done once, prior to or concurrent with the receipt of A's first message from B. Once B's certificate is authenticated it need not be re-authenticated until it is revoked, or expires. This has obvious performance advantages when B communicates frequently with A.

A number of digital certificates exist in S-BGP. One is issued by the authority responsible for allocating IP address space. This is used to authenticate that an AS (named in the certificate) has the authority to announce that it originates a prefix (also named in the certificate). Another is issued by the authority responsible for allocating AS numbers, binding organization names with AS numbers; yet another is issued to bind a public key to an AS number, and still another authenticates a given speaker to represent a given AS number, binding a public key to that authorization. The public keys associated with this last type of certificate are the ones most frequently used in the course of operations. Without going into detail, for the purposes of optimizing route authentication we will assume—as in previous S-BGP analyses—that all the keys necessary to support S-BGP can be distributed as needed (and infrequently) to the speakers that need them.

Now consider how announced routes are authenticated. Each speaker has its own key pair, and is able to obtain the public key of any other speaker. Without security, a speaker s_i in B_i would forward to a speaker s_{i-1} in B_{i-1} an empty claim that a prefix p in B_k is reached through a $B_i - B_k$ path. Instead, speaker s_i performs a hash on the sequence $B_{i-1}, B_i, \ldots, B_k, p$, signs that hash value using its private key, and appends that signature to a list of such signatures generated by previous speakers on the AS path. Speaker s_i sends the AS sequence and the signature sequence in one message to s_{i-1}. This message (and the outer signature) means that s_i has a nice path to p via a $B_{i+1} - B_k$ path, and is offering to extend this $B_i - B_k$ path to B_{i-1}. The signature will be used to validate that s_i is the author of this message. The next inner signature attests that s_{i+1} offered to extend the $B_{i+1} - B_k$ path to B_i; and so on.

Conversely, when s_{i-1} receives this message, it can use s_i's public key to verify that s_i really appended itself to an AS path supposedly from s_{i+1}, and that s_i authorizes s_{i-1} to use the extended AS path. Speaker s_{i-1} can only accept the path supposedly from s_{i+1} by using s_{i+1}'s public key to verify that it did indeed append itself to a path supposedly from s_{i+2}. Speaker s_{i-1} must then verify s_{i+2}'s signature, and so on. [1] If all the signatures verify, then s_{i-1} can conclude that the path was was not forged, that at least one point in time it was constructed legitimately under BGP rules.

Our work is motivated by the fact that when a speaker forwards an **Update** message to a set of peers, each peer receives a slightly different message, therefore requiring that each message be individually signed. Each such message contains the identity of the receiver, because for any given **Update** the speaker may, or may not include a given peer in the set of recipients. Failure to include the recipient's identity in the message leaves open the possibility of a "cut-and-paste" attack, wherein the attack *observes* an **Update** message sent by s_i to a peer, and sends a copy to another of s_i's peers that was not in the recipient set.

Murphy [21,20] considers additional issues and countermeasures for network-level integrity and replay attacks, such as using IPSec, and the MD5 option on the TCP level [10]. The cascade of digital signatures in route announcements, coupled with authentication of the binding between AS and prefix, is the centerpiece of S-BGP, and is the object of our attention.

4. Cryptographic overhead

If we want to protect against a malicious router forging paths, but we do not want to change the basic method that BGP routers use to distribute and accumulate path information, then it would seem that the cascaded signatures used by S-BGP are unavoidable. Each router in a claimed path must attest to its participation with a digital signature. The question then arises of how much this security costs in terms of performance.

RSA is the near-universal standard for public-key cryptography (although, for many years, the US Government did not recognize it). In RSA, the key pair consists of a pair of exponents and a large (typically 1024-bit) modulus. In a private-key operation, we raise the input to the power of the private exponent, and reduce it via the modulus; in the public-key operation, we use the public exponent.

The RSA key generation process lets us choose a value for one of the exponents; since the time of an RSA operation is roughly proportional to the position of the leading one in the exponent, we typically choose the public [2] exponent to be very

[1] In S-BGP there is appended a short index to a certificate, assumed to be known to the recipient, that binds speaker, AS and speaker's public key. This certificate is used to identify the speaker's AS and validate that it is authorized to sign for that AS.

[2] If we instead chose a small value for the private key, then it would be easy for the adversary to guess it.

small (such as 17). Together, these properties enable signature verification in RSA to be much quicker than signature generation.

Since RSA was patented, and since RSA could be used for encryption as well as signatures, the US Government promulgated an alternative public-key scheme, DSA, which could be used for signatures only. DSA does not share the property that verifications are extremely quick.

In both RSA and DSA, special large integer parameters play important roles. The security depends on the presumed intractibility of certain operations on these large integers; consequently, the longer the integers, the more secure.

In both schemes, these parameters relate to a modulus whose length has practical size impacts. First, the length of a signature will be the length of a fixed number of integers between 0 and the modulus. Second, the maximum length of the message operand is also the length of the modulus. Since messages in general will be much longer, signature schemes in practice use cryptographic *hash* functions, as noted earlier. A cryptographic hash function h transforms an arbitrary length message to a fixed-length hash value, with the property that it is believed infeasible for an adversary to calculate another message that transforms to that same value. For signatures with hashing, one first hashes the message, and then uses public-key cryptography on the hash value. Currently, the standard hash function is *SHA-1*, which generates hash values 20 bytes long.

Currently, 1024-bits is considered the shortest reasonable modulus for RSA, giving 128 bytes as the signature length. 1024 bits is also considered the shortest reasonable value for the p parameter in DSA, with a modulus of 160 bits and a signature size of 40 bytes.

We benchmarked these operations by using the OpenSSL [1] library (version 0.9.6.d), on a 1 GHz PC running RedHat 7.2 Linux. We then normalized these figures for a CPU speed of 200 MHz, as the estimates we made for normal BGP **Update** processing were taken from a router with that clock rate. The actual CPU speed in our study is largely immaterial; we are interested in the relationship between numbers, not the numbers themselves. Furthermore, the assumed communication latencies between speakers is not large enough to contribute greatly to simulation timing.

It is in theory possible to break up the DSA signature operation into two steps, one of which may be done before the message to be signed is known. The requirements for this *pre-computation* include (naturally) that the pre-computed values be saved in a secure fashion to reduce the potential for an adversary to use them to forge signatures. Perhaps because of this, we were not able to find a public domain crypto package that supports [3] pre-computation. Nevertheless, if a problem domain is important enough it is reasonable to assume that the effort will be made to take advantage of this feature. Correspondingly we decomposed and bench-marked the OpenSSL implementation of DSA to measure signature times under the assumption that pre-computation is employed. The cost of signing virtually disappears. Tables 1 and 2 show these results.

[3] Crypto++ has an exponentiation trick it calls "pre-computation," but this is not the same thing.

Table 1
Benchmarks for RSA and DSA algorithms, based on a 1024-bit RSA modulus and a 1024-bit DSA p parameter

Operation type	RSA sign	RSA verify	DSA sign (no p-c)	DSA sign (p-c)	DSA verify
Time (1 GHz) (ms)	10.0	0.5	5.1	0.003	6.2
Time (200 MHz) (ms)	50.0	2.5	25.5	0.015	31.0

Table 2
Benchmarks for SHA-1 operations

Data size (bytes)	1–56	57–64	65–120	121–128
SHA-1 time, 1 GHz (μs)	1.57	2.74	2.58	3.76
SHA-1 time, 200 MHz (μs)	7.87	13.71	12.90	18.82

The time needed for hashing is proportional to the length of the data size, stepping up linearly in accordance with the SHA-1 construction.

S-BGP uses DSA, because of its shorter key length, and the potential to exploit pre-computation in the signature step [13]. The flip-side though is that verifications take an order of magnitude longer in DSA than in RSA, and a path suffix is verified potentially many times, while it is it signed but once. We will shortly revisit this tradeoff.

The use of cryptography adds an overhead of CPU cycles for each **Update** message, both for verification and for signing. The values of Table 1 in the 200 MHz need to be considered in light of measurements we made of **Update** processing costs on a BGP speaker at Dartmouth College. [4] We observed **Update** costs ranging from 32.5 to 97.5 ms. Table 1 shows then that cryptographic overhead is very significant. Prior work recognized this as a potential problem, and proposed some methods for improvement. Murphy [20] suggested limiting the number of signatures that need to be verified; instead of verifying each signature in the path, only work on up to c signatures. This limits the cost of verifying paths, but sacrifices some security. Other solutions involve caching routes; we discuss those in more length in the following section.

5. Simulations

In order for us to evaluate the impact that **Update** processing under S-BGP might have on convergence, we use simulation. The complexity of interactions in BGP make it difficult to analytically predict the effect of cryptographic overhead on convergence, the fact that convergence is a global property means that to measure it in the wild one has to deploy S-BGP on a large scale. Simulation is the obvious—and

[4] This device has a Cisco RSP4 processor of 200 MHz with 128MB memory. It runs Cisco IOS 12.2(3), maintaining 5 configured peers. The routing table holds roughly 120,000 entries. This is a typical configuration for a router at the edge of the network.

only—tool for this kind of “what-if” problem. Our experiments use the SSFNet [3] simulator, which has been used in a number of other BGP studies [17,6,22,36,19]. This simulator has a large number of options for configuring BGP behavior, and we use the defaults, such as

- the policy for selecting a path is shortest-number-of-hops,
- no router is a reflector,
- no route aggregation is performed,
- default timer values are used. The most important of these for convergence studies is MRAI.

Related BGP simulators include Genesis [31] and BGP++ [25]. Genesis is based on SSFNet but uses an entirely different approach to management of synchronization in a parallel/distributed context. BGP++ puts a simulation wrapper around an actual BGP implementation, Zebra [12].

Our experiments were conducted on three topologies, representing 110, 512, and 715 ASes. The process we used to create the topology is more detailed than it is interesting, the essential thing it strives to preserve is distribution of connectivity in the interior of the Internet. We start with a large AS topology built from Internet measurements, heuristically merge nodes to reduce the graph, then reduce the graph further by randomly removing edges and retain the largest connected component, and then merge low degree nodes some more. While admittedly heuristic, the graphs we obtain share the heavy-tail-like distribution of connectivity seen in the real Internet. The cummulative distribution function of our 715 AS model is illustrated in Fig. 1. Here we see that while the maximum node degree is 164, the average degree is 7.6. Indeed, nearly 80% of ASes have degree smaller than the average, while 10% have degree larger than 20. The core of the Internet is a highly connected place.

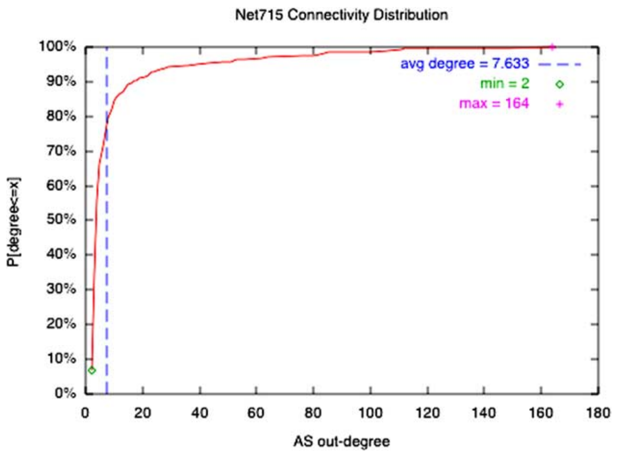

Fig. 1. Degree distribution of 715 AS system.

In our experiments every node represents an AS. We make the simplifying assumption that each AS originates 2 prefixes and that each AS is represented by just one BGP speaker. Nevertheless, these graphs will suit our purposes well enough. Our objective is to determine whether (and under what conditions) route verification might impact BGP convergence, and our topologies should give insight into that question. Another objective is to see to what extent optimizations we design reduce cryptographic overhead, and our topologies should tell us that too. They reflect the connectivity of ASes at the core, they reveal the impact of their high load and high degree of connectivity, and they serve to generate the density of alternative routes that can stress BGP.

Our simulation model applies execution delays to **Update** processing at three different stages. First, every **Update** received by a router has a nominal service time sampled uniformly between 32.5 and 97.5 ms. This range is based on measurements taken on a local BGP router, with a 200 MHz CPU clock. Route filtering and route selection are functions whose execution costs are included in this delay. Another execution delay for verification follows, if and only if the route is chosen as the basis for an announcement. *Thus all of our experiments verify only those routes which must be verified.* Following this delay, if an announcement is to be made, a set of **Update** messages for the router's peers are generated and placed in an output buffer. The cost of generating digital signatures for these is incurred at this point. The messages are sent after they are signed, and when the MRAI timer permits.

We designed a set of experiments on the 110 AS model to examine convergence under S-BGP as a function of BGP load intensity. In one scenario we measure the time required for routes to a newly announced prefix to completely propagate through the network. The originating router announces its two prefixes to its peers, at a time when no router has an entry in its forwarding tables for either prefix. The announcement wave that follows establishes, at every router, routes to these prefixes. In a second scenario we model a router rebooting after a crash, and measure the time needed for all routes to all prefixes to converge. The volume of workload is much higher, because a rebooting router will get table dumps—an announcement for each prefix, from each of its peers. It will announce its own preferences for prefixes as it processes all of these **Updates**. In addition, the rebooting router announces its own two prefixes, which thus entails all of the work involved in the first scenario as well.

In both scenarios we initiate the experiment at three different routers: the one with highest connection (24), one with median connection (6), and one with smallest connection (2). We ran each of the six resulting experiments twenty times, and compute the mean values of a variety of measures. The ratio of standard deviation to mean is less than 5% throughout, and so for our purposes it suffices to report the means.

For each experiment we report the number of route announcements, the number of **Update** messages (a count that includes withdrawals), the number of times a signature was verified, the number of times a signature was generated, the sum over all routers of the CPU time allocated to nominal **Update** processing, the sum over all routers of the CPU time spent in cryptographic related activities, and the convergence time. For S-BGP we considered DSA with pre-computation support for signatures (pDSA), and ordinary DSA.

Table 3
Single prefix insertion experiment

Protocol	#Anns.	#**Updates**	#verif.	#sigs.	base CPU (s)	crypto CPU (s)	Convergence (s)
24-Peer Route Announces							
BGP	578.8	649.6			42.5		74.0
S-BGP (pDSA)	584.8	651.7	942.4	560.8	42.2	29.2	74.3
S-BGP (DSA)	582.7	649.7	947.2	558.7	42.5	43.6	74.3
6-Peer Router Announces							
BGP	633.5	731.4			47.7		81.5
S-BGP (pDSA)	599.6	685.9	1321.4	593.6	44.4	40.9	81.5
S-BGP (DSA)	610.3	703.2	1383.0	604.3	45.6	58.2	81.0
2-Peer Router Announces							
BGP	653.7	763.0			49.5		87.9
S-BGP (pDSA)	644.0	745.8	1660.8	642.0	48.4	51.5	86.8
S-BGP (DSA)	647.6	750.0	1654.4	645.6	48.7	67.7	87.9

pDSA denotes DSA with aggressive pre-computation.

Cryptography does not not affect convergence in these experiments, as seen in Table 3, despite the fact that cryptography increases the cost of processing an **Update** by 50–140%, depending on cryptography used, and connectivity of the announcing router. [5] In this case route propagation is limited by the MRAI timers, rather than the **Update** processing time.

Connectivity reflects how close a node is to the center of the graph. Convergence time increases as connectivity decreases because the AS paths involved are longer. DSA cryptography costs increase with decreasing connectivity for the same reason.

Things are more interesting when we consider convergence under the high load induced by a rebooting router, shown in Table 4. Here we also include experiments that assume RSA is used.

One worry might be that in this scenario convergence time is nothing more than the time the rebooting router needs to process the table dumps it gets from its peers. There are 218 prefixes that originate in other ASes, and each peer reports its path to each. This means that the highly connected router receives $24 \times 218 = 5232$ **Updates**; with an average of 65 ms. nominal processing per update, it takes 340 s (of the 472 s convergence time) to work through the table dumps; in the medium connected case it takes 85 of 150 s, in the least connected case it takes 28 of 100 s. Table dump processing is an important component of convergence time, but is not the only component.

To understand this table it is helpful to consider a simple model of the cost of processing a received **Update** message:

$$F + \Pr\{\text{route preferred}\} * (L * C_v + N_p * C_s)$$

[5] The seeming inversion of BGP and S-BGP convergence values in the 2-Peer subtable is not statistically significant. The two values involved, 87.9 and 86.8, are within each other's 95% confidence intervals.

Table 4
Simulated behavior of rebooting convergence experiment

Protocol	#Anns.	#**Updates**	#verif.	#sigs.	base CPU (s)	crypto CPU (s)	Convergence (s)
24-Peer Router Reboots							
BGP	28559.3	33220.5			2157.3		472.4
S-BGP (pDSA)	29324.0	34044.9	52626.8	29420.0	2212.3	1632.4	629.9
S-BGP (DSA)	29711.9	34438.8	52870.0	29807.9	2238.2	2399.6	799.3
S-BGP (RSA)	29061.5	33766.6	52194.6	29157.5	2194.0	2707.2	793.4
6-Peer Router Reboots							
BGP	4182.4	4760.8			309.4		150.9
S-BGP (pDSA)	4214.0	4802.7	7263.0	4238.0	311.9	225.3	232.6
S-BGP (DSA)	4251.3	4843.3	7309.8	4275.3	314.7	335.7	261.3
S-BGP (RSA)	4249.7	4855.4	7423.0	4273.7	315.6	232.2	199.5
2-Peer Router Reboots							
BGP	1950.3	2266.5			147.1		110.2
S-BGP (pDSA)	1955.9	2280.5	1494.9	512.7	148.1	59.7	112.6
S-BGP (DSA)	1983.9	2308.5	5389.8	1991.9	150.1	167.1	115.3
S-BGP (RSA)	1964.9	2277.4	5208.4	1972.9	147.8	111.6	115.0

pDSA uses aggressive pre-computation for signatures; DSA does not. We also simulated S-BGP with RSA, for comparison.

where F is a fixed cost, L is length of the AS path, C_v is the cost of verifying a signature, Pr{route preferred} is the probability that the **Update** reports an AS path that the recipient prefers, N_p is the number of peers receiving the resulting **Update**, and C_s is the cost of signing that **Update**. For pDSA C_s is cheap but C_v is expensive; for RSA the roles are reversed. DSA's value for C_s is much more significant than pDSA's. Thus for pDSA the term $\Pr\{\text{route preferred}\} * L * C_v$ is the critical added cost, while for RSA it is $\Pr\{\text{route preferred}\} * N_p * C_s$.

We see that in the highly connected router experiments, RSA's crypto costs are much larger, and convergence time is much longer than ordinary BGP. The same is true for ordinary DSA. In the medium connected case, far fewer peers are sent **Updates** when a new route is advertised. This significantly reduces the average cost of processing an **Update** using RSA. The impact of pDSA and DSA on convergence is still notable. However, convergence is hardly affected by cryptography in the low connectivity case.

Our data suggests that under high load, cryptographic operations can degrade BGP convergence. This observation is consistent with the experiments by Premore and Griffin [6] who observed that convergence time tends to increase as the cost of processing an **Update** increases. Our model of increased costs is somewhat more complex than theirs, but the result seems to hold at high load. Crypto's extra computational load affects convergence when the network can least afford degradation—under high load, such as has been induced during worm attacks or route flapping. This concern is echoed in [13], where a number of options for caching are suggested:

- Cache validated routes on received **Update** messages, to avoid the cost of re-validation.
- Cache signed **Updates** sent to peers, to avoid the cost of re-signing a previously announced route.
- Save these caches in non-volatile memory, to avoid the overwhelming cost of verifying announcements at reboot.

To this list we add the idea of caching individual AS Path suffixes (as opposed to just route announcements), which would allow efficient verification of a route announcement that has not been seen before, but for which a route with a common suffix has.

The degree to which such caching is practical, or effective (in the case of caches too small to remember *every* route or suffix) depends a great deal on available memory, and traffic patterns. To get some feeling for the demands on such a system, we analyzed a BGP announcement feed from a RIPE [2] monitor (peer rrc00) for the entire month of March 2002. We observed 2,092,981 unique suffixes, and 1,143,903 unique routes. S-BGP already exerts a heavy memory footprint, as it needs to store extracts of certificates; we would like to avoid exacerbating additional memory demands. Nevertheless, for the purposes of finding an upper bound on network performance we ran the rebooting experiments under the assumption that all of the caching mentioned above is in effect. These results are shown in Table 5. We report again the ordinary BGP behavior, for reference. We use the prefix 'c' to remind us that caching is assumed.

Under these idyllic conditions, the volume of cryptographic overhead is dramatically reduced over that of Table 4, where caching is not assumed. The very interesting fact though is that this massive reduction had a comparatively smaller impact on convergence time. The increase in convergence using cDSA or RSA is still large. However, it stands to reason that if a router can remember *every* route, then if we

Table 5
Rebooting convergence experiment

Protocol	#Anns.	#**Updates**	#verif.	#sigs.	base CPU (s)	crypto CPU (s)	Convergence (s)
24-Peer Router Reboots							
BGP	28559.3	33220.5			2157.3		472.4
S-BGP (cpDSA)	29063.1	33754.7	3997.5	9537.4	2193.7	127.3	496.1
S-BGP (cDSA)	28900.8	33560.0	3383.6	9063.5	2180.6	339.2	648.7
S-BGP (cRSA)	29069.9	33750.8	3098.1	8961.9	2191.5	459.5	782.6
6-Peer Router Reboots							
BGP	4182.4	4760.8			309.4		150.9
S-BGP (cpDSA)	4230.5	4825.9	1846.5	1551.9	313.3	57.7	180.6
S-BGP (cDSA)	4210.5	4791.3	932.6	1535.5	311.4	68.5	187.0
S-BGP (cRSA)	4229.8	4828.3	1844.9	1572.2	313.8	83.8	201.1

cpDSA uses aggressive pre-computation for signatures and aggressive caching; cDSA uses standard DSA and aggressive caching; cRSA uses standard RSA and aggressive caching.

run the simulation long enough every route will have been seen and no further cryptography need be done, so there is a limit to what we can infer from this experiment. A real study of caching performance is beyond the scope of this paper.

Despite the limitations of simulation, we believe one can conclude from our experiments that if routes (and suffixes) are not cached, then under high load and significant connectivity, the cryptographic overhead of securing route announcements can adversely affect BGP convergence. Whether optimizations such as caching and DSA pre-computation can avert that threat is an open question whose answer may not be known until actual deployment. In the following section we propose an optimization that offers the hope of achieving nearly the convergence of ordinary BGP, but without the overhead and uncertainty of caching.

6. Signature amortization

Questions related to reducing the cost of cryptography in the routing context have been raised before, e.g. [9,35]. Such methods typically work to reduce the cost by reducing the dependence on public-key methods, i.e. develop different ways of authentication. As useful as line of approach like these may be, there are real difficulties in applying those methods in BGP. The approach we explore is to do expensive private-key operations less often, amortizing that cost over multiple messages.

As we have seen, the most significant drawback of DSA is its high validation cost. If we eschew caching, then we cannot escape validating every path suffix when we validate an AS path. From this point of view RSA is more attractive, because its validation cost is (by our measurements) 12.5 times faster. Where RSA fails us in this context is the high cost of signing every **Update**.

Recall the reason for the signature explosion: when a speaker makes an announcement, it makes it to multiple peers (potentially). In BGP the messages to peers are identical; in S-BGP they are not. Security requires that the recipient be named and be part of the message that is signed. Thus, at first glance, every message must be signed individually.

6.1. Amortization across peers

Let us take a second glance and ask ourselves whether there is not a way to achieve the same security. There is a disarmingly simple solution. Suppose that a speaker logically enumerates its peers with indexes ranging from 0 to the maximum number of peers, N. In practice $N < 64$. A speaker can thus use a bit-vector, N bits long, to describe any subset of its peers. The idea then is to have a speaker create a bit-vector that describes the full set of peers filtered to receive an **Update**, put the bit-vector in the message rather than the recipient's identity, and sign the message. *This one message can be sent to all the peers, and only one new signature is involved.* If a speaker knows its logical position in a peer's enumeration, it can validate that an **Update** was intended for it by simply checking whether its bit is set. We call this optimization *Signature-Amortization* (S-A).

The only issue left is determining how a speaker learns its logical identity in each of its peer's enumerations, and how it can prove this identity to its relying parties. A direct solution involves PKI certificates. Recall that in the S-BGP framework a speaker acquires the certificate of each peer, principally to obtain its public key. Certificate formats are general enough that we can require that a speaker's certificate name each of the speaker's peers (e.g., in an extension). We simply require that this naming include the enumeration.

For a speaker s_i to prove that it was a recipient of an update from s_{i+1}, it would need to show that s_i is the kth peer of s_{i+1}, where the kth bit in the vector (signed with the update) was set. But if this (k, s_i) pair is in s_{i+1}'s certificate, then attestation of this information is already available to any party that can verify the signature.

This solution does require generation of new certificates when peers change, but that is a relatively infrequent event. As described it does have the vulnerability that one of s_i's peers can determine from certificates and bit vectors who other of s_i's peers may be. This vulnerability, and a dependence on certificates could be addressed by other methods that involve direct communication between peers.

Amortization of the same message to multiple peers will have the biggest impact—obviously—at a highly connected router. Most routers do not have the connectivity found in Tier 1 ASes, and so we ask whether it is possible to find another way of amortizing a private-key signing. The solution is to aggregate the signature on all messages in all buffers, while the router awaits the firing of a MRAI timer.

6.2. Amortization across output buffers

We can adapt the hash tree techniques of [18,24] to this problem. Operationally what we do is to tag **Update** messages that are going into an output buffer as being "unsigned". These messages will contain bit-vectors, reflecting a cross-peer aggregation that we continue to exploit. We delay actual signature until the message is free to be transmitted—either immediately, or (if it must wait for its MRAI timer to fire) when *any one* of the MRAI timers fires. At that point we "sign" all of the messages in all of the buffers that are tagged as "unsigned", and change the tag in each. The messages that are released by the MRAI timer are sent (possibly leaving some signed messages in other buffers, but they will not be signed again). The advantage to the hash-tree method is that we can sign all of these messages using just one expensive private-key operation. (The advantage to also using bit-vectors is that the hash-tree need be built using only one representative of the group of **Updates** resulting from the same announcement.)

More generally, given a set $\mathscr{M}$ of messages, our goal is to produce a signature for each one, that has the same properties as traditional signatures: e.g., $s(m)$ is a detachable blob matching m, that could only have been produced when A intended to sign m, and that can be verified using m, $s(m)$, and knowledge of A's public key (and, in particular, *not* requiring some other set of previous signatures). However, we want to minimize the number of private key operations necessary to go from $\mathscr{M}$ to $\{\langle m, s(m)\rangle | m \in \mathscr{M}\}$.

Let h be a sufficiently strong cryptographic hash function; let $\circ$ denote concatenation; let L, R denote a simple encoding of "left" and "right", and let *pk_sign* and *pk_verify* be some standard public-key signature scheme. Suppose $\mathscr{M}$ consisted of 2 different messages, m_L and m_R; then we could apply the private key to obtain

$$S = pk\_sign(h(h(m_L) \circ h(m_R)))$$

We could then use $S \circ h(m_R) \circ R$ as the signature on m_L; verification consists of hashing the message, concatenating $h(m_R)$ on the right, hashing the result, and verifying that S is the public-key signature. In general, for a set of 2^k messages, we could sign them by building a binary tree of depth k and doing a private-key operation on the root; the "signature" of any given message consists of the private-key signature of the root, along with the path from that message to the root, specified via k pairs of hashes and L, R values.

More formally, for a set $m_1, \ldots, m_K$ of messages, we define a hash tree of this set to be a (directed) binary tree with K leaves. We label each leaf with an $h(m_i)$; if an interior node has children labeled N_L and N_R, then we label that node with $h(N_L \circ N_R)$.

Let $Root(m_1, \ldots, m_K)$ be the label on the root of such a tree.

If r labels the root of the tree and N is the label on some node, we define the $Route(N, r)$—the route of N to r—as follows: If $N = r$, then $Route(N, r) = \emptyset$ (trivially—we're already there).

Otherwise, N must be an interior node. Let N_s be its sibling and N_p be its parent. We then define the remaining cases:

$$Route(N, r) = \begin{cases} (N_s, R), Route(N_\mathrm{p}, r) & \text{if } N \text{ is the left child} \\ (N_s, L), Route(N_\mathrm{p}, r) & \text{if } N \text{ is the right child} \end{cases}$$

The intuition here is that (N_s, R) describes a step in the path: "concatenate N_s to the right of the current hash value, hash that pair, and keep going."

With these definitions, we can define the signature for a message m_i with a bit vector V, using a tuple of values. The first value is

$$pk\_sign(Root(m_1, \ldots, m_K)),$$

a digital signature on the root of the hash tree. The remaining values are the bit vector V and the list

The remaining values are the list

$$Route(h(m_i), Root(m_1, \ldots, m_K)),$$

which describes the route from $h(m_i)$ at a tree leaf to the root, including the hash values needed at every level of the tree to reconstruct $Root(m_1, \ldots, m_K)$. Validation of this signature is tantamount to doing exactly that—use the path information to reconstruct $Root(m_1, \ldots, m_K)$, and verify that that is what the speaker for B_i signed. Observe that of all the messages in the buffers that result from the same announcement, each receives the same signature, and only one instance of that common message is used to build the hash-tree.

Fig. 2 illustrates these ideas with a simple example. At the time an MRAI timer fires, a router has unsigned messages for two peers, in two different buffers. A hash

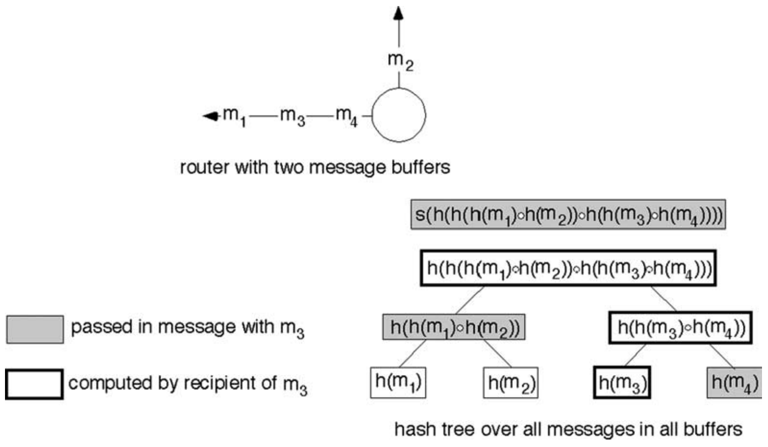
m2
m1 — m3 — m4
router with two message buffers
s(h(h(h(m1)∘h(m2))∘h(h(m3)∘h(m4))))
h(h(h(m1)∘h(m2))∘h(h(m3)∘h(m4)))
h(h(m1)∘h(m2))
h(h(m3)∘h(m4))
h(m1)
h(m2)
h(m3)
h(m4)
passed in message with m3
computed by recipient of m3
hash tree over all messages in all buffers

Fig. 2. Illustration of hash-tree, and associated fields sent with m_3, and recomputed by m_3's recipient.

tree is built, the leaves of which are hashes on the individual messages. The value assigned to an interior node is the hash of the concatenation of the values of its children. The figure illustrates what accompanies m_3 when it is sent (in addition to an indexing code which identifies the position of $h(m_3)$ in the tree) and the values that are recomputed by the receiver. The basic idea is that the receiver recompute all the values on nodes on the path from the message's place in the leaves to the root, and that the message contain all the "extra" hash information needed to support those computations. The receiver then verifies that the root value it reconstructed is the root value that was signed. It is clear then that the amount of information comprising a signature grows logarithmically in the size of the tree, as does the work of verifying the message.

6.3. Performance

We have identified two ways of amortizing the cost of a private-key signature for BGP announcement processing. We call this *Signature-Amortization* (S-A). If we couple this technique with RSA, we obtain the benefits of fast verification, with the advantage of reduced signing cost. Table 6 shows this, where for reference we include the behavior of ordinary BGP and of the highly optimized cpDSA approach. S-A-P reflects the behavior when the signature is amortized only over the peers, S-A-B reflects the behavior when the signature is amortized over buffers.

Now we see that the impact that RSA has on convergence is small. In the case of the highly connected router, this small device reduces the fraction of overall processing time dedicated to crypto from 55% to 16%, with a reduction of convergence time from 168% of BGP's to just 105% of it. It is interesting to observe that amortizing over buffers *and* peers is not appreciatively better than amortizing over peers alone. It turns out that in these experiments the average size of the hash tree is pretty small—less than 2.5 in both cases, while the average size of a peer set is 7.3 in the

Table 6
Rebooting convergence experiment

Protocol	#Anns.	#**Updates**	#verif.	#sigs.	base CPU (s)	crypto CPU (s)	Convergence (s)
24-Peer Router Reboots							
BGP	28559.3	33220.5			2157.3		472.4
S-BGP (cpDSA)	29063.1	33754.7	3997.5	9537.4	2193.7	127.3	496.1
S-A-P (RSA)	28731.9	33392.4	51584.4	5840.3	2170.9	420.9	497.2
S-A-B (RSA)	28595.1	33252.8	51385.6	4342.8	2160.6	345.8	493.2
6-Peer Router Reboots							
BGP	4182.4	4760.8			309.4		150.9
S-BGP (cpDSA)	4230.5	4825.9	1846.5	1551.9	313.3	57.7	180.6
S-A-P (RSA)	4269.0	4857.9	7356.8	981.3	315.8	67.4	164.4
S-A-B (RSA)	4243.3	4833.3	7283.0	905.7	313.9	63.5	162.7

S-A-P uses signature amortization across peers: we sign common messages resulting from the same **Update** message only once. S-A-B also uses signature amortization across buffers: we build a hash-tree on messages in buffers awaiting release by MRAI timers.

highly connected case, and 5.8 in the medium connected case. It is not difficult to imagine situations where the hash-trees will be larger though. The value of the MRAI timer is pretty standard, and does not seem to be changing in practice while routers get faster. So, for example, if a router had a 1 GHz CPU rather than the 200 MHz CPU we assume, the potential exists for buffers to have 5 times more messages in them when MRAI timers fire, because the CPU can move them there that much faster.

Of course, the most significant aspect of these results is that signature aggregation appears to deliver the same—or better—convergence as does the highly optimized S-BGP, but without pre-computation, and without caching.

The additional memory demands of S-A include an 88-byte increment per AS in a path for using RSA rather than DSA, and $20 \log K$ more bytes per AS for a hash-tree of height K. Based on a 4096 byte limit on **Update** size, there is still ample room for our more complex signatures. Our calculations (based on the analysis in [13]) is that if the average hash-tree is built on as many as 32 messages, 21 ASes can still fit in on **Update** message. Should the 4096 byte barrier be immutable, one could alter the hash-tree amortization to trigger as soon as the number of unsigned messages reached a threshold value. For aggregation across peers, the added cost of the bit-vector is inconsequential, and in any case can replace explicit identification of the recipient.

7. Parallelization

The topology of the 110 AS model studied in the previous section models the connectivity of the Internet in and near the core. However, the diameter (longest shortest path between any two ASes) is 5 hops, which is quite a bit shorter than typical paths in the Internet. This model cannot be expected to capture convergence effects in the

Internet due to path distance. Because of the rate-limiting accomplished by the MRAI timer, (30 s per hop), distance effects could be significant. However, in order to retain the connectivity structure of the Internet *and* better reflect path-length effects, we need to run experiments on larger models.

SSFNet's BGP simulator provides a highly detailed model of BGP operations. In particular, every BGP speaker contains a forwarding table unique to it, capable in principle of routing to any advertised prefix in the entire model. Furthermore, BGP calls for a router to retain the last path advertised to every prefix, by every one of a router's peers. Consequently the number of AS paths the simulator must retain as state is proportional to the product of the number of routers, the number of unique prefixes, and the average number of peers per router. As we increase the size of models simulated by increasing ASes, we increase both the number of routers and the number of prefixes (because new ASes advertise new prefixes), and we tend to increase the average AS path length (thereby increasing the storage needed per AS path). Furthermore, the means by which we increase model size tends to increase connectivity as well. Consequently the memory demands on our simulator grow by more than a square in the number of ASes modeled. It is well known that virtual memory is not an effective way to increase the size of simulation models, as discrete-event simulations lack the locality of reference upon which virtual memory depends. Other means are needed.

One way of acquiring a large memory space is to run the simulation on a distributed memory parallel computer. We have recently completed an implementation of the SSFNet simulator that runs on such platforms. We used this implementation to study SBGP and S-A on a model with with 512 ASes, and another with 715 ASes.

The experiments already reported on the 110 AS model looked at convergence when a single router reboots. In order to explore the impact of cryptographic overhead in other contexts, we developed experiments that cause a set of sessions to fail within a 60-s interval. Routers themselves do not fail, only connections between routers. Failure of a session will cause both endpoints to advertise new paths to any prefixes whose previously announced paths used that session; recovery of the session can cause another wave of advertisements that effectively restore the paths lost when the session failed. In our experiments we randomly choose a set of sessions, and failed them at scattered times in a 60-s window (after all startup advertisements have converged). Roughly 30% of all sessions are reset. This experiment models situations where abrupt congestion in a network affects the TCP sessions between routers that do not have physical connections (such as in a peering hotel).

7.1. BGP convergence

Table 7 presents the results from the larger runs, using the same notation as employed for the 110 AS model experiments.

Notable points in these data include

- The added processing delays of S-BGP increases the number of announcements and updates over ordinary BGP, while acceleration methods (S-BGP with cach-

Table 7
Reset sessions convergence experiment

Protocol	#Anns.	**#Updates**	#verif.	#sigs.	base CPU (s)	crypto CPU (s)	Convergence (s)
512 ASes, session drops							
BGP	1779K	2043K			132,805		1538
S-BGP (DSA)	1851K	2213K	13,806K	1918K	143,873	428,135	4638
S-BGP (cpDSA)	1740K	2013K	4109K	1728K	130,898	128,090	1848
S-A-B (RSA)	1801K	2068K	13,534K	36K	13,4457	35,722	1729
512 ASes, session returns							
BGP	1261K	1276K			80,962		1155
S-BGP (DSA)	1315K	1328K	6536K	1423K	85,420	20,1739	3008
S-BGP (cpDSA)	1323K	1335K	785K	397K	85,926	24,526	1232
S-A-B (RSA)	1247K	1263K	6166K	26K	80,666	16,663	1256
715 ASes, session drops							
BGP	5982K	7102K			461,643		3077
S-BGP (DSA)	10,171K	11,899K	93,187K	11,094K	773,446	288,9687	9996
S-BGP (cpDSA)	6193K	7354K	13,552K	5913K	478,024	422,617	4741
S-A-B (RSA)	6649K	7997K	60,547K	123K	519,835	157,852	4503
715 ASes, Session Returns							
BGP	1808K	1816K			117951		1629
S-BGP (DSA)	2347K	2652K	18,395K	1907K	171,839	569,645	4988
S-BGP (cpDSA)	1855K	1863K	729K	356K	121,139	22,971	1610
S-A-B (RSA)	1755K	1764K	8378K	30K	114,161	22,469	1583

S-BGP DSAd uses standard S-BGP with DSA; cpDSA uses S-BGP with aggressive DSA pre-computation and aggressive caching. S-A-B uses RSA and signature amortization across peers and buffers.

ing, and S-A) have announcement and update counts that are much closer to BGP's.
- The raw convergence time increases with model size.
- The relative differential between BGP and S-BGP convergence times increases with model size.

These tendencies might have been anticipated without resorting to the simulation. However, the magnitude of the relative differences would not have been, nor the magnitude of the convergence times that result from these experiments. A 10K second convergence time (715 ASes, S-BGP) is nearly 3 h—that is a significant finding that one would have difficulty predicting without using simulation. It is a finding that could be obtained only by exploiting the large memory resource of a distributed memory multiprocessor.

7.2. Parallel performance

The large-scale experiments just reported were conducted on a distributed memory cluster, using 4–16 nodes, with two 2.8 GHz CPUs and 4Gb memory on each node. We evaluate performance by dividing the AS network into 32 "timelines" (threads that coordinate in simulation time with each other), and considering per-node performance on 4, 8, and 16 nodes, where nodes simulate (respectively) 8, 4, and 2 timelines each. The AS graph is divided somewhat evenly among the 32 timelines, although it is very difficult to balance workload precisely because the workload is so dynamic.

This implementation of SSFNet synchronizes timelines by imposing a barrier synchronization every Δ units of simulation time, where Δ is the smallest latency delay on any communication between any two timelines. $\Delta = 200$ ms in these experiments. The current distributed memory implementation of SSFNet uses Java's sockets for all communication. This is an inherently non-scalable approach, because a socket connection must be established between every unique pair of timelines. Future releases of the package will offer improvements that employ faster non-socket communication mechanisms.

Fig. 3 illustrates the distribution of time in a 512 AS experiment between three activities. "send" reflects time spent sending messages between nodes, "sync" reflects time spent at a barrier synchronization at the end of a synchronization step, and "exc" reflects everything else, presumably model execution time. We do not see much speedup going from 4 to 16 nodes, the overall execution time drops from just under 1200–800 s. Two factors seem to dominate these performance results. One is load imbalance, evidenced by variation in the fraction of running time allocated to "exc". This is more explicitly illustrated in Fig. 4, where for each node we plot the number of events executed by the node by the average number of events executed by a node. A value of 1.0 means the node hits perfect balance. In the 16 node experiment node 8 is 28% underloaded, while node 4 is 38% overloaded. One of the challenges of load-balancing for a problem like this is that the location of the workload is highly variable, and largely unpredictable. A second factor degrading performance is that synchronization overhead per node tends to go up as we increase the number of nodes. This is to be expected, however this cost is exacerbated by our reliance on sockets. The combination of decreasing workload per node with increasing synchronization cost means that performance gains of distributing workload are significantly diminished.

The memory required for the 512 AS experiments is a little over 4Gb in total, or just 1Gb per node on the 4 node runs and 250Mb per node on the 16 node runs. It is possible to equip a desktop computer with this much memory, although that is costly. Experiments on the 715 AS model use over 10Gb; it is again possible (but unusual) to equip a single CPU with this much memory. Our attempts to evaluate an 830 AS model on 16 nodes encountered memory problems, solved simply by using more nodes in the cluster. In this region of model space we clearly encounter the size limitations imposed by faithful replication of BGP state data on every BGP speaker. However, the ability of SSFNet to use multiple computer's memories on a single problem opens the way to run experiments on much larger models than are possible using a normal desktop computers.

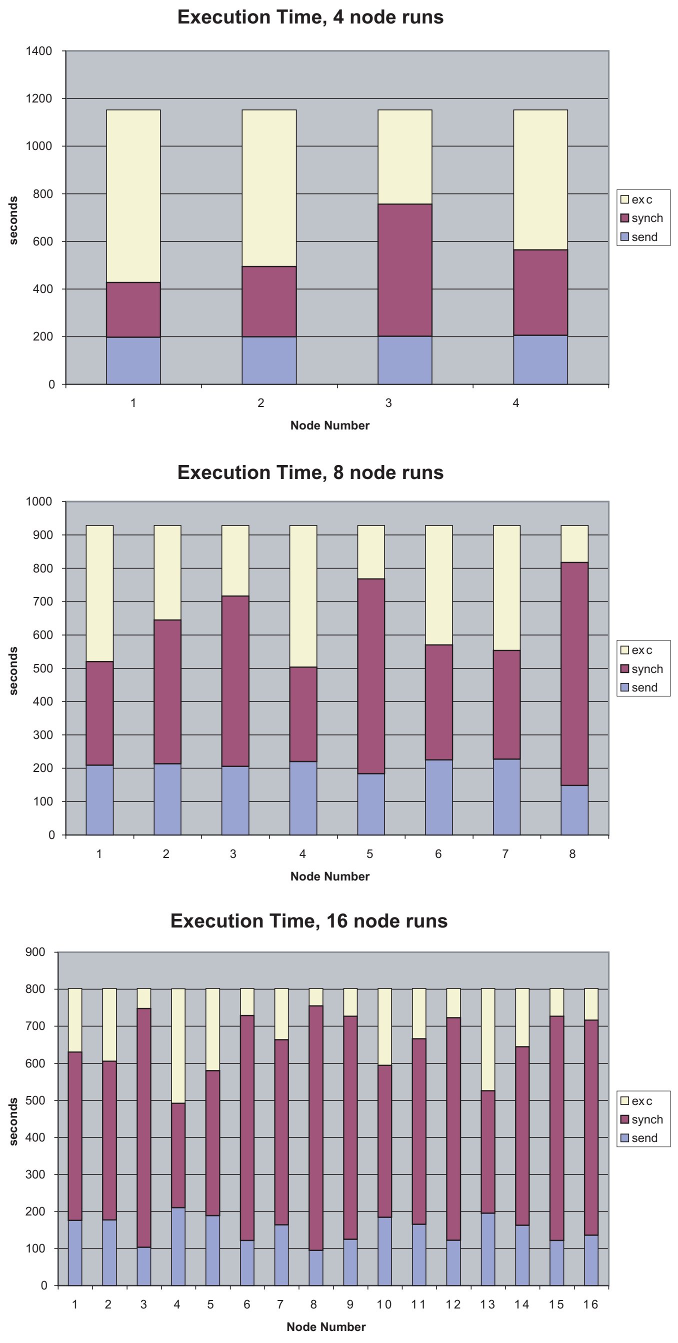

Fig. 3. Distribution of execution time in parallel simulation of 512 AS network.

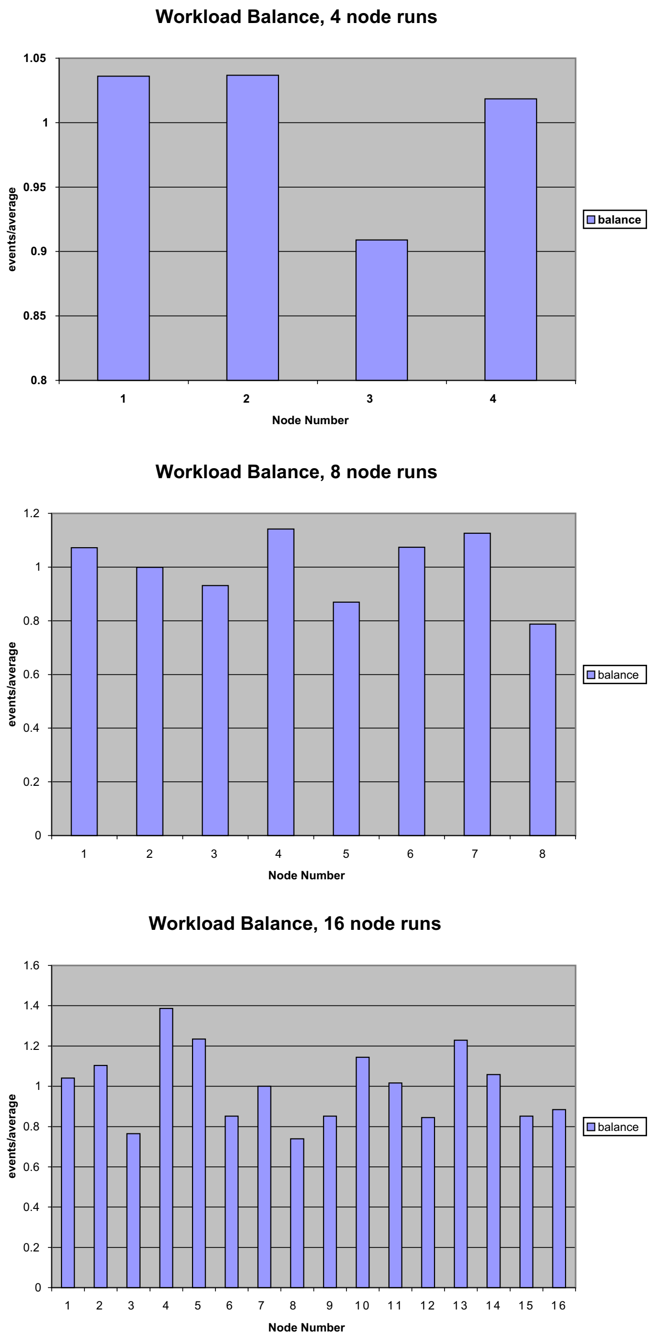

Fig. 4. Distribution of event executions in parallel simulation of 512 AS network.

8. Reflections

The *real* BGP routing infrastructure is enormous, and our simulations have had to make a number of simplifying assumptions. It is therefore important to reflect on how the differences in assumed models and real systems might affect the results.

A very useful aid for reasoning about what is going on is the graph shown in Fig. 5. This captures the sense of observations others have made [23] on how convergence (the y-axis) behaves as a function of the default MRAI timer (x-axis). For the first part of the curve it is decreasing convex, after which it becomes increasing and linear. The graph is understood if one thinks of BGP announcements as arriving in "waves", the release of which is governed by the MRAI timer. When the timer is so large that all the announcements in a wave are processed before the timer fires again, then the growth of convergence time is linear in the MRAI value because between waves the system is simply waiting for the "go" signal to release the next wave. This is the second part of the curve. The first part of the curve is explained by the realization that holding announcements back allows for a "better" new announcement for a prefix to overwrite a buffered-but-as-yet-unreleased suboptimal one. As the MRAI timer decreases in length, the chance *increases* that multiple updates for the same prefix are released, and generate more work. The convergence time is larger for small MRAI values, because there are more announcements being processed.

Understanding this leads us to the curve shown in Fig. 6. The y-axis is as before; the x-axis is something like router utilization. Recalling our notion of a wave of announcements (when the MRAI is large enough to cause each router to idle between waves), we divide the amount of computational work in a wave by the MRAI value, assumed in this graph to be fixed. Value 1 (or 100%) is saturation. Prior to saturation the convergence time is unaffected by workload, because it is defined by the MRAI value—this corresponds to the increasing linear portion of the curve in Fig. 5. The curve beyond the saturation point corresponds to the convex part of

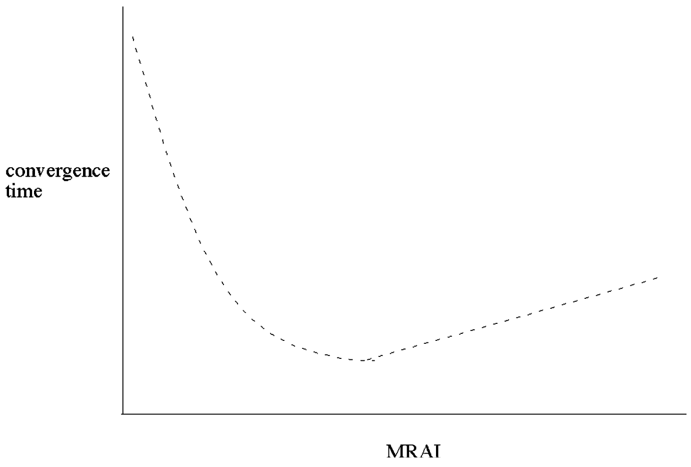

Fig. 5. Convergence time as a function of MRAI value.

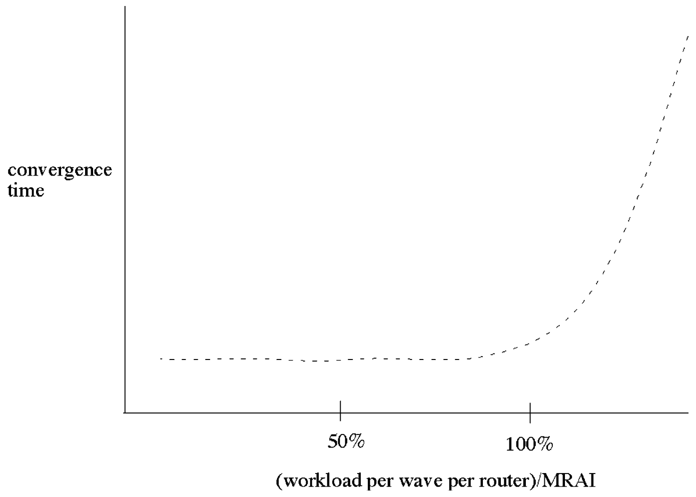

Fig. 6. Convergence time as a function of router workload.

Fig. 5, but in reverse. The insight is that in saturation conditions it is equivalent to talk in terms of decreasing MRAI and increasing workload per announcement—it is merely a matter of selecting measurement units.

With this understanding we can analyze the impact that our model assumptions have on the comparison of BGP, S-A, and S-BGP. The key is to see that our simplifications uniformly underestimate the amount of work done by a router per wave. Larger networks will have longer AS paths and higher average connectivity. Both factors increase cryptographic workload processing, in the former case because verification costs increase linearly in the AS path length, and in the latter case because signature costs (in S-BGP) increase linearly in the number of peers receiving an **Update**. Our assumption of two announced prefixes per AS underestimates the number of **Updates** processed per wave in the real Internet. Let u_0, u_1 and u_2 be the values on Fig. 6 corresponding (respectively) to BGP, S-A, and S-BGP in a simulation run, and let u'_0, u'_1 and u'_2 denote the same values in the real Internet. Let c be the function name. We want to know how the different approaches compare in the real Internet, with respect to the comparison in our simulations: how does $c(u'_1) - c(u'_0)$ compare with $c(u_1) - c(u_0)$, $c(u'_2) - c(u'_1)$ compare with $c(u_2) - c(u_1)$, and $c(u'_2) - c(u'_0)$ compare with $c(u_2) - c(u_0)$?

The first thing we notice is that any increase in workload for the pure BGP case is inherited by both other cases—if BGP's workload increases by a factor of $\alpha > 1$, then S-A's and S-BGP's workload increases by factors at least as large as α. This implies that $u'_0 - u_0 \leqslant u'_1 - u_1$ and $u'_0 - u_0 \leqslant u'_2 - u_2$. Thus, because c is increasing convex, its derivative $\dot{c}$ is positive and increasing, which implies that

$$\begin{aligned} c(u_1) - c(u_0) &= \int_{u_0}^{u_1} \dot{c}(s)\mathrm{d}s \leqslant \int_{\alpha u_0}^{\alpha u_1} \dot{c}(s)\mathrm{d}s = \int_{u'_0}^{\alpha u_1} \dot{c}(s)\mathrm{d}s \leqslant \int_{u'_0}^{u'_1} \dot{c}(s)\mathrm{d}s \\ &= c(u'_1) - c(u'_0). \end{aligned}$$

The derivation for $c(u_2) - c(u_0)$ is identical. It follows that the raw increase in convergence time due to cryptographic overhead suggested by our simulations will be even larger in the actual Internet.

It is also useful to consider the comparison of S-A and S-BGP. Compared to the cost of a signature, the added complexity of verifying a signature with S-A schemes is small. On the other hand, the real Internet will have larger peer groups than do our simulations, hence S-BGP will be adding signature costs that S-A avoids. This implies that if u'_1 is larger than u_1 by a factor of β, then u'_2 is larger than u_2 by a factor at least as large as β. A derivation identical to the one above then shows that $c(u_2) - c(u_1) \leqslant c(u'_2) - c(u'_1)$, which is to say that in the real Internet the S-A optimizations will be even more effective than our simulations suggest.

9. Conclusions

The Border Gateway Protocol is the glue of the Internet, but it is vulnerable to attack. Public-key cryptography can help protect it, but is computationally expensive and may impact network performance. We ask whether the route validation technique in S-BGP can affect BGP convergence. Using a detailed simulation model, we find that the answer is "yes", when the BGP processing load is high in a router that has many peers.

Minimizing performance impact of standard S-BGP requires caching and pre-computation, which create implementation difficulties. Pre-computation requires maintenance of a protected cache of pre-computed values. Furthermore, one needs the cache the most, when the ability to replenish it is least—when a router has a heavy load of **Updates**. We considered very aggressive caching strategies for S-BGP, and assumed that once a route was seen, it was remembered forever. Under these assumptions we see that S-BGP convergence is close to that ordinary BGP. However, the effectiveness any *real* caching strategy depends heavily on the pattern of traffic, on the replacement policy, and on the amount of memory available for the cache. Our simulations suggest that S-BGP, as proposed, has to find effective solutions to the caching and pre-computation problems.

We approach the problem differently. We notice that RSA has a much lower validation cost than DSA, but that its signing cost is overwhelming. We develop methods for amortizing that expensive private-key signature over many updates. Our simulations show that by basing the cryptography on RSA and amortizing the signature cost, BGP convergence is as good or better than the highly optimized S-BGP solution—but without the complications, uncertainties, and risks of that solution. It is possible therefore to minimize the impact route validation has on convergence, simply by being careful with signatures.

Another approach to reducing cryptographic costs might be to sprinkle trusted witnesses throughout the net. These witnesses could verify the cascaded signatures on a path, then replace this cascade with a single signed assertion, and possibly apply even more amortization there. In 1994, [29] suggested using secure coprocessor hardware to implement such witnesses for a different type of multiparty hearsay; since

1997, strong programmable secure coprocessor platforms have been available as COTS products [30]. We also plan to explore the performance impacts of this approach on large systems, using simulation. On a different note, recent work [11] explores new schemes for securing routing information based on symmetric cryptography; further exploration and simulation could be interesting.

Our work is unique in its assessing BGP security measures in terms of its impact on BGP convergence time, and is unique in using parallel simulation techniques to study that impact on larger models than can be studied on normal serial computers. The experiments on large models we report on suggest two importance facts. The first is that S-BGP convergence time after a major disruption of sessions is very large, on the order of 3 h in our of our experiments. The other is that the performance impact of the cryptographic optimizations we develop improve as the size the network grows. This observation suggests that the optimizations are worth studying in the largest network of all, the actual Internet.

For simulations such as ours, portability (and hence access by the larger simulation community) is an important consideration. A strength of our simulator is that it is built entirely using Java—even the communication/synchronization aspects of our distributed version use core Java functionality. Hence the software forming the basis of our experiments is highly portable.

Acknowledgements

The authors are grateful to Guy Blelloch, Steve Campbell, Michael Liljenstam, and B.J. Premore for their helpful suggestions. This project was supported under Award No. 2000-DT-CX-K001 from the Office for Domestic Preparedness, U.S. Department of Homeland Security. Points of view in this document are those of the author(s) and do not necessarily represent the official position of the U.S. Department of Homeland Security. Smith was also supported in part by the Mellon Foundation and AT&T/Internet2; Smith and Nicol were supported in part by NSF Grants CCR-0209144 and EIA-98-02068. Nicol is supported in part by by DARPA Contract N66001-96-C-8530, and Department of Energy contract DE-AC05-00OR22725. Accordingly, the US Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for US Government purposes.

References

[1] OpenSSL: The Open Source toolkit for SSL/TLS. Available from <http://www.openssl.org>.
[2] Ripe: Réseaux IP Européns. Available from <http://www.ripe.net>.
[3] SSFNet: Scalable Simulation Framework—Network Models. http://www.ssfnet.org. See http://www.ssfnet.org/publications.html for links to related publications.
[4] N. Doraswamy, D. Harkins, IPsec: The New Security Standard for the Internet, Intranets, and Virtual Private Networks, Prentice Hall, 1999.

[5] R. Govindan, A. Reddy, An Analysis of internet inter-domain topology and route stability, in: Proceedings of INFOCOM 1997, April 1997, pp. 850–857.
[6] T.G. Griffin, B.J. Premore, An experimental analysis of BGP convergence time, in: Proceedings of ICNP 2001, November 2001, pp. 53–61.
[7] T.G. Griffin, F. Bruce Shepherd, G. Wilfong, Policy disputes in path-vector protocols, in: Proceedings of ICNP 1999, October 1999, pp. 21–30.
[8] T.G. Griffin, G. Wilfong, An analysis of BGP convergence properties, in: Proceedings of SIGCOMM 1999, August 1999, pp. 277–288.
[9] R.C. Hauser, T. Przygienda, G. Tsudik, Lowering security overhead in link state routing, Computer Networks 31 (8) (1999) 885–894.
[10] A. Heffernan, RFC 2385: Protecting of BGP Sessions via the TCP MD5 signature option, 1998.
[11] Y. Hu, A. Perrig, D. Johnson, Efficient security mechanisms for routing protocols, in: Proceedings of Network and Distributed System Security Symposium, San Diego, California, February 2003.
[12] K. Ishiguro, G. Zebra. <http://www.zebra.org>.
[13] S. Kent, C. Lynn, K. Seo, Secure Border Gateway Protocol, IEEE Journal of Selected Areas in Communications 18 (4) (2000).
[14] C. Labovitz, A. Ahuja, A. Bose, F. Jahanian, Delayed Internet routing convergence, in: Proceedings of SIGCOMM 2000, August 2000, pp. 175–187.
[15] C. Labovitz, A. Ahuja, F. Jahanian, Experimental study of internet stability and wide-area backbone failures, in: Proceedings of the International Symposium on Fault-Tolerant Computing, June 1999.
[16] C. Labovitz, A. Ahuja, R. Wattenhofer, S. Venkatachary, The impact of internet policy and topology on delayed routing convergence, in: Proceedings of INFOCOM 2001, April 2001, pp. 537–546.
[17] Z. Morley Mao, R. Govindan, G. Varghese, R.H. Katz, Route flap damping exacerbates internet routing convergence, in: Proceedings of SIGCOMM 2002, August 2002.
[18] R. Merkle, Protocols for public key cryptosystems, in: Proceedings of 1980 Symposium on Security and Privacy, IEEE Computer Society, April 1980, pp. 122–133.
[19] J. Joyce Mulligan, Detection and Recovery from the Oblivious Engineer Attack, Master's thesis, Massachusetts Institute of Technology, September 2002.
[20] S. Murphy, BGP Security Protections, Internet-Draft, draft-murphy-bgp-protect-00.txt, NAI Labs, February 2002.
[21] S. Murphy, BGP Security Vulnerabilities Analysis, Internet-Draft, draft-murphy-bgp-vuln-00.txt, NAI Labs, February 2002.
[22] D. Pei, X. Zhao, L. Wang, D. Massey, A. Mankin, S. Felix Wu, L. Zhang, Improving BGP convergence through consistency assertions, in: Proceedings of INFOCOM 2002, June 2002.
[23] B. Premore, An Analysis of Convergence Properties of the Border Gateway Protocol Using Discrete Event Simulation, PhD thesis, Dartmouth College, June 2003.
[24] R. Merkle, A certified digital signature, in: G. Brassard (Ed.), Advances in Cryptology—CRYPTO'89, Lecture Notes in Computer Science, vol. 435, Springer-Verlag, 1990, pp. 218–238.
[25] G. Riley, BGP++. <http://www.ece.gatech.edu/research/labs/MANIACS/BGP++/>.
[26] A. Shaikh, A. Varma, L. Kalampoukas, R. Dube, Routing stability in congested networks: experimentation and analysis, in: Proceedings of SIGCOMM 2000, August 2000, pp. 163–167.
[27] B. Smith, J.J. Garcia-Luna-Aceves, Efficient security mechanisms for the Border Gateway Routing Protocol, Computer Communications 21 (3) (1998) 203–210.
[28] B. Smith, S. Murthy, J.J. Garcia-Luna-Aceves, Securing distance vector routing protocols, in: Proceedings of Internet Society Symposium on Network and Distributed System Security, San Diego, California, February 1997.
[29] S. Smith, D. Tygar, Security and privacy for partial order time, in: ISCA Seventh International Conference on Parallel and Distributed Computing Systems, October 1994.
[30] S. Smith, S.H. Weingart, Building a high-performance, programmable secure coprocessor, Computer Networks 31 (April) (1999) 831–860.
[31] B.K. Szymanski, Yu Liu, R. Gupta, Parallel network simulation under distributed genesis, in: Proceedings of the Seventeenth Workshop on Parallel and Distributed Simulation, June 2003.

[32] H. Tangmunarunkit, R. Govindan, S. Shenker, D. Estrin, The Impact of routing policy on internet paths, in: Proceedings of INFOCOM 2001, April 2001, pp. 736–742.
[33] I. van Beijnum, BGP: Building Reliable Networks with the Border Gateway Protocol, O'Reilly, 2002.
[34] K. Varadhan, R. Govindan, D. Estrin, Persistent Route Oscillations in Inter-Domain Routing, Technical Report 96-631, USC/Information Sciences Institute, March 1996.
[35] K. Zhang, Efficient protocols for signing routing messages, in: Symposium on Network and Distributed Systems Security (NDSS '98), San Diego, California, 1998.
[36] X. Zhao, D. Pei, L. Wang, D. Massey, A. Mankin, S. Felix Wu, L. Zhang, Validation of the MOAS conflicts through assertions, in: Proceedings of the International Conference on Dependable Systems and Networks, June 2002.