NetReview: Detecting when interdomain routing goes wrong

Andreas Haeberlen[†⋆] Ioannis Avramopoulos[◇] Jennifer Rexford[‡] Peter Druschel[†]

[†] *Max Planck Institute for Software Systems (MPI-SWS)* [‡] *Princeton University*
[⋆] *Rice University* [◇] *Deutsche Telekom Laboratories*

Abstract

Despite many attempts to fix it, the Internet's interdomain routing system remains vulnerable to configuration errors, buggy software, flaky equipment, protocol oscillation, and intentional attacks. Unlike most existing solutions that prevent specific routing problems, our approach is to detect problems automatically and to identify the offending party. Fault detection is effective for a larger class of faults than fault prevention and is easier to deploy incrementally.

To show that fault detection is useful and practical, we present NetReview, a fault detection system for the Border Gateway Protocol (BGP). NetReview records BGP routing messages in a tamper-evident log, and it enables ISPs to check each other's logs against a high-level description of the expected behavior, such as a peering agreement or a set of best practices. At the same time, NetReview respects the ISPs' privacy and allows them to protect sensitive information. We have implemented and evaluated a prototype of NetReview; our results show that NetReview catches common Internet routing problems, and that its resource requirements are modest.

1 Introduction

Global Internet connectivity is the result of a competitive cooperation of tens of thousands of Autonomous Systems (ASes) using the Border Gateway Protocol (BGP). Unfortunately, interdomain routing is plagued with many serious problems: BGP is hard to manage, and BGP misconfigurations and software bugs can create severe network disruptions [8, 24, 37]. Equipment failures in one AS can cause route flapping and trigger excessive routing announcements in ASes many hops away [35]. The inadvertent configuration of conflicting routing policies in a collection of ASes can lead to persistent oscillation [14]. An adversary that controls a BGP-speaking router can intentionally 'hijack' another AS's address block in order to discard the data packets, snoop on the traffic, impersonate the legitimate destination, or send spam [25, 27].

Many (but not all) of these problems are rooted in the absence of a mechanism to verify routing information. BGP essentially allows anyone to announce any route, whether that route actually exists or not. Hence, there has been a lot of work on securing BGP. However, most of this work focuses on *fault prevention*, that is, masking routing problems by suppressing invalid route announcements. This approach is effective against many common problems, but it cannot prevent other, equally common faults; for example, an ISP might fail to announce a route because of an incorrect export filter. Existing security extensions to BGP, such as S-BGP [22] and soBGP [34], are not effective against such faults. Moreover, existing fault prevention systems require significant buy-in before they can yield much benefit, and they require an Internet-wide public-key infrastructure (PKI); for these and other reasons, prevention systems have not yet achieved widespread deployment.

In this paper, we take a different and complementary approach, namely *fault detection*. If we cannot prevent every routing problem, why not at least ensure that each problem is detected and linked to the ISP that caused it? Fault detection is easy to deploy incrementally: it does not require a central PKI or cryptography on the critical path, and it yields benefits even when the deployment consists of just a few ISPs (or even a single ISP). Moreover, if we accept the possibility of some delay between the occurrence of a fault and its detection, we can catch a very general class of faults, including router and link failures, software bugs, misconfigurations, policy violations, and even attacks by hackers or spammers. In particular, we can detect faults that would be difficult or impossible to prevent, e.g., when a faulty or misconfigured router fails to propagate certain routes.

Fault detection has two main benefits. The first (and most obvious) benefit is that ISPs are automatically informed about routing problems and their causes, which enables them to respond quickly. Thus, ISPs no longer have to rely on monitoring heuristics or customer complaints to find out about problems, which increases cus-

tomer satisfaction and enables ISPs to swiftly respond even to minor problems. Also, since detection links faults to their causes, ISPs no longer need to diagnose faults manually. Finally, ISPs obtain a 'safety net' that enables them to respond to unexpected problems.

The second, more indirect benefit of fault detection is that it makes an ISP's reliability transparent. Today, ISPs may have little to gain from pushing reliability beyond a certain point, since customers cannot easily attribute a given routing problem to a particular ISP. Fault detection is an opportunity for reliable ISPs to showcase their good performance and to distinguish themselves from the competition, which could help them attract new customers. In the long term, this could even result in a market for reliability, in which customers could directly compare the routing performance of potential providers.

At first, fault detection may appear to be a simple matter of keeping logs of all routing messages and inspecting them (perhaps even manually) for routing problems. However, the problem is complicated by several unique aspects of the interdomain routing system. First, detecting certain types of faults requires that ISPs share information, because the fault cannot be detected based on one ISP's view of the network alone. However, ISPs wish to minimize the amount of information they release to their competitors. Thus, a detection system must balance its detection power against the scope of the information ISPs need to release. Second, the amount of log data collected is so vast that manual inspection is out of the question, except in the most egregious cases. Third, the logs may be incomplete or even incorrect, not least because the routing system is often attacked by hackers who may try to manipulate records in order to cover their tracks. Finally, if the information about faults is to be used as a measure of reliability, we must avoid both false positives and false negatives, which rules out heuristic solutions.

To demonstrate that fault detection is viable, we present NetReview, a system that implements fault detection for BGP. NetReview reliably and automatically detects routing problems by checking secure traces of BGP messages against high-level specifications of the expected routing behavior. NetReview respects the ISPs' privacy and provides strong guarantees: it does not produce false positives or false negatives even when under attack by a Byzantine adversary. Using a prototype implementation of NetReview, we show that its resource requirements are modest, and that it is effective against common Internet routing problems.

Existing work on securing interdomain routing has proven difficult to deploy. A natural question is whether a fault detection system would be hampered by similar problems. To address this concern, we show that NetReview can overcome common deployment hurdles: it can work with existing router hardware, it does not require a global PKI, it can be deployed incrementally, and it offers immediate benefits to early adopters.

The rest of this paper is structured as follows. In Section 2, we begin by giving some background on BGP, and we discuss the specific challenges of BGP fault detection. In Sections 3 and 4, we present the design of NetReview and its specification language. In Section 5, we report results from a feasibility study to show that fault detection is practical. In Section 6, we present solutions to various deployment-related problems, such as operation in a partial deployment or without a CA, and we point out incentives for adoption by ISPs. In Section 7, we describe some advanced features that could be added to NetReview. Section 8 discusses related work, and Section 9 concludes this paper.

2 Background

2.1 Interdomain routing with BGP

The Internet consists of independent administrative entities called *autonomous systems (ASes)*. An AS usually corresponds to a network run by an Internet Service Provider (ISP), although some large ISPs have multiple ASes. Each AS is assigned a unique *AS number (ASN)*; in 2008, about 40,000 ASNs were in active use. In addition, each AS owns a set of IP addresses, which it can assign to its hosts and routers. Usually, ASes use large contiguous sets of addresses that share a common *prefix*; for example, the prefix `128.42.0.0/16` covers all IP addresses whose first two octets are 128 and 42.

To exchange routing information with each other, all ASes use the *Border Gateway Protocol* [28]. Each AS designates some of its routers as *BGP speakers*, which are then connected to BGP speakers in adjacent ASes. When a BGP speaker learns of a route to a new prefix, it can *announce* that route to its peers in adjacent ASes; if the route becomes unavailable later, it must *withdraw* the announcement. BGP is a path-vector protocol, that is, each announcement contains the sequence of ASes that the route traverses in an attribute called `AS_PATH`.

BGP specifies a mechanism for exchanging routing information. Which routes to use and whether or not to announce them to peers is decided independently by each AS according to its own *policy*; for example, an AS might prefer short routes to reduce latency. Some aspects of the policy are determined by an AS's business relationships; for example, an AS might agree to act as the *provider* for another AS, and it would then be expected to offer its customer a route to every prefix it can reach. Adjacent ASes usually sign a *peering agreement*, which specifies the obligations of each peer.

2.2 What is a BGP fault?

The specification of BGP in RFC 4271 [28] describes a message format and a few basic rules; everything else is left to the implementation and the policy of an AS. Therefore, we use a very generic definition of a BGP fault. Suppose we have a complete message trace M_a of all BGP messages a given AS a has sent or received over time (both internally and to/from its peers). Then we simply assume that there is a deterministic function $F_a(M_a, t)$, and we say that AS a is *faulty* at time t if and only if $F_a(M_a, t) =$ true, otherwise we say that AS a is *correct* at time t.[1] Note that F_a is specific to AS a; a different AS b could have a different function F_b.

How can such a function F_a be defined? There are several sources of information that can be used for this purpose (of course, multiple sources can be combined):

- **RFC 4271:** The AS is faulty if it violates the BGP specification, e.g., by sending a malformed message, or by announcing a path that contains a loop.
- **ASN and prefix assignment:** The AS is faulty if it uses a foreign AS number, or if it announces a prefix it does not own.
- **BGP best practices:** The AS is faulty if it does not follow current best practices, e.g., by failing to aggregate prefixes correctly.
- **Peering agreements:** The AS is faulty if it does not honor the peering agreements it has negotiated with its peers, e.g., by failing to export its customers' routes, or by choosing a route through an AS it has promised to avoid.
- **Connectivity:** The AS is faulty if it fails to offer routes to certain prefixes, e.g., because an internal link or equipment failure has caused a partition.
- **Internal goals:** The AS is faulty if its routers fail to achieve some goal the AS has set for itself, e.g., by choosing an expensive route over a cheaper one due to a configuration error.

Note that our definition does not say who defines F_a and who evaluates it; we will address these challenges in Section 3, and we will show which information needs to be shared to ensure that faults are detected. Also, our definition does *not* imply that there is a unique correct message trace for each AS. For example, if an AS is offered multiple routes to a given prefix and its policy does not prefer any route in particular, it can choose any route.

According to our definition, each fault is local to a single AS. Thus, if a faulty AS a exports a bad route to a neighbor b, b does *not* become faulty for propagating the route – except if propagating the route constitutes a fault according to its own function F_b. A special case occurs when a link between two neighboring ASes fails. Since the link is shared by two ASes, we cannot attribute this event to an individual AS, so we attribute it to the pair of ASes instead.

[1] A similar definition can be used for router-level faults. We focus on AS-level faults because they are more general.

2.3 Challenges in BGP fault detection

To illustrate the challenges in building a practical BGP fault detection system, we first consider a simple strawman implementation of fault detection that works as follows. Every ISP enables full logging on all their routers and periodically uploads the logs to a central server, together with a description of their peering agreements and internal goals. Because the central server has full information, it can reconstruct the message trace M_a for each AS a, and it can evaluate F_a for any (past) point in time. This solves the fault detection problem because the central server can eventually detect *any* BGP fault, no matter how complex it is.

However, there are several reasons why this strawman solution would not work in practice:

- **Privacy:** The strawman's logs contain sensitive information that ISPs would not agree to reveal to a third party, such as their routing policy and internal topology. A practical system must protect the ISPs' business secrets while retaining its detection power.
- **Reliability:** The information in the strawman's logs is not necessarily accurate: routers can malfunction, and hackers can tamper with the logs to conceal an attack. A practical system must ensure that no faults go undetected, even when it is under attack.
- **Automation:** Collecting and processing the vast amounts of trace data could prove expensive. A practical system must be able to efficiently check this data without manual intervention.
- **Decentralization:** It is unlikely that ISPs around the world would accept and trust a single fault detector entity. A practical system must not introduce any new trusted entities or require ISPs to coordinate with ISPs they do not already cooperate with.
- **Deployability:** The strawman assumes global deployment. A practical system must have a clear deployment path, with immediate benefits for early adopters and a migration path for legacy equipment.

A fault detection system for BGP should address these five challenges.

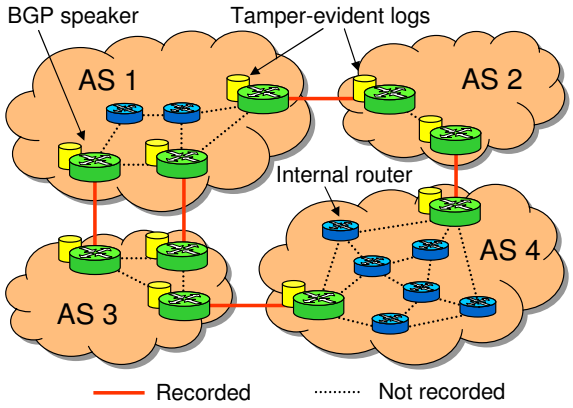

Figure 1: System model. Each BGP speaker maintains a tamper-evident log of the BGP messages it exchanges with other ASes. Internal routing messages are not recorded.

3 NetReview

To demonstrate that the above challenges can be addressed in a practical system, we now present a detection system called *NetReview*. For clarity of presentation, we initially assume that NetReview is deployed universally, and that the allocation of ASNs and IP prefixes to ASes is certified by a trusted certification authority (CA). In Section 6, we describe solutions for partial and incremental deployment, and we show how NetReview can be used without a CA.

3.1 Overview

At a high level, each BGP speaker maintains a log of all the BGP messages it sends and receives (Figure 1). In addition, each AS states a set of *rules* that describe best practices, routing policies, etc. that the AS adopts (the union of these rules specify F_a and thus define what constitutes a fault; they do not necessarily describe the entire routing policy of the AS). Both the logs and the rules are then made available to certain other ASes, who can *audit* them to check whether the rules have been followed. If a rule was broken, NetReview guarantees that at least one auditor can detect this and obtain *verifiable evidence* of the fault, which it can then use to convince third parties.

NetReview only records BGP messages that are exchanged with other ASes, but no internal routing messages. Thus, the log only contains information that an AS would reveal to other ASes anyway; the ISP's proprietary information, such as its internal topology, is not revealed. In addition, each ISP is free to decide which rules it wants to reveal to each auditor. For example, an ISP might choose to reveal its best-practice rules to everyone, and, in addition, it might reveal to each of its business partners a set of rules that describes its policy towards that partner. This is safe because the partner already knows that aspect of the policy from the peering agreement.

NetReview uses cryptographic authenticators [17] to detect if routing messages are not logged correctly. The log itself is tamper-evident, that is, it can detect if log entries are modified after the fact. Thus, NetReview can guarantee that log corruption – due to software bugs or hardware malfunctions – cannot cause faults to go undetected. This guarantee holds even in the presence of Byzantine faults, e.g., when hackers or spammers attempt to cover up the traces of an attack.

NetReview includes a simple specification language for writing rules. The resulting rules can be checked efficiently; we show that a commodity workstation is sufficient to audit several ASes in real time.

NetReview is designed to leverage existing trust and business relationships between *neighboring* ASes. We consider two ASes to be neighbors if they are connected by a direct link.

3.2 Assumptions and guarantees

NetReview's design relies on the following assumptions:

1. **Each AS has at least one diligent neighbor.** By diligent, we mean that this neighbor regularly audits the AS and collects evidence. This is a reasonable assumption because ASes have a natural interest in learning about routing problems of their neighbors.

2. **Each AS is willing to publish a list of its neighbors.** Knowing the nature of the business relationships is not necessary, just the fact that two ASes are connected. This is a reasonable assumption, because the information can already be determined using tools like traceroute or RouteViews [30].

3. **Each AS can eventually send control messages to any other AS.** This property holds for the Internet because the AS graph is connected, and because link failures are repaired in a timely fashion (that is, within at most a few days).

4. **No attacker can invert the hash function or break cryptographic keys.** This is a common assumption for protocols that rely on cryptography.

Note that NetReview is not subject to the limitations for Byzantine fault tolerance techniques, such as the need for $3f + 1$ replicas to tolerate f faults. Fault detection is an easier problem, so this bound does not apply.

NetReview focuses on detecting *observable* faults, that is, faults that a) causally affect at least one non-faulty AS [16], and b) violate a rule that is revealed to at least one diligent AS. This restriction is inevitable because we

cannot expect faulty routers to help with fault detection. An example of an unobservable fault would be two faulty routers sending bad routing updates to each other, but neither of them logging the messages or forwarding the authenticators to the other's neighbors. Such a fault cannot be detected as long as it does not affect a correct AS.

Under the above assumptions, NetReview guarantees that a) any observable fault is eventually detected and irrefutably linked to a faulty AS, and that b) no verifiable evidence is ever generated against a non-faulty AS.

3.3 Maintaining tamper-evident logs

In NetReview, each border router maintains a log of all routing messages it has sent to, or received from, a router in another AS. In addition, the logs contain periodic checkpoints of the BGP routing tables, as well as a hash of each rule the AS has adopted. This additional information is needed for auditing and will be discussed in Sections 3.8 and 3.9, respectively.

NetReview's logs are based on the logs in PeerReview [17]. The logs are *tamper-evident*, that is, a router either records precisely the messages it has exchanged with other routers, or it is possible to detect that the router is faulty. Note that, since our goal is fault detection, we do not need to prevent faulty routers from tampering with their logs – being able to detect tampering is sufficient because it is clear evidence of a fault. Specifically, NetReview detects if a router (i) records a message it did not send or receive, (ii) omits a message it did send or receive, (iii) changes an existing log entry, or (iv) keeps multiple logs or a branched log. For lack of space, we only sketch the most important aspects of the log here. Please refer to [17] for a complete description.

Operation: Each log is structured as a hash chain, i.e., every entry e_i is associated with a sequence number s_i and a hash value h_i that covers the entry itself and, transitively, all the previous entries. To explain the protocol for logging message exchanges, we use the example of two routers, Alice and Bob. Whenever Alice sends a message m to Bob, Alice first appends a `SEND(m)` entry to her log and then attaches an *authenticator* to m, which is a signed statement that Alice has logged the transmission of m. The authenticator $\alpha_i = \sigma_{Alice}(s_i, h_i)$ for an entry e_i includes the entry's value in the hash chain h_i and is signed with Alice's cryptographic key σ_{Alice}. The authenticator has two purposes: first, it convinces Bob, and any auditors of Bob's log, that the message is authentic, which rules out (i). Second, it serves as evidence that a `SEND(m)` entry must appear in Alice's log, which addresses case (ii) and, because of the hash chain, case (iii). When the message m arrives, Bob appends a `RECV(m)` entry to his log and then returns an acknowledgment to Alice, which includes an authenticator for the `RECV(m)` entry. At this point, both Alice and Bob have obtained evidence that the other side has properly recorded the message in their log.

NetReview imposes a limit on the number of unacknowledged messages that can be in flight between Alice and Bob at any given time. If this limit is reached, e.g., during an unplanned physical link failure or because Bob refuses to send acknowledgments, the operators are notified and must resolve the problem by leveraging their existing business relationship.

What if Alice or Bob log the message at first but modify or remove it later? When Bob receives the authenticator from Alice, he detaches it from the message (to save bandwidth) and forwards it to Alice's neighbors. Thus, Alice's neighbors eventually learn of all log entries for which Alice issued authenticators. Each neighbor periodically inspects Alice's log to check whether these entries actually appear. If an authenticator is properly signed but the corresponding entry is missing, then Alice must have tampered with the log, maintained multiple logs or a log with multiple branches, and the authenticator is a signed confession. This addresses (iv).

Protocol support: NetReview extends BGP with support for authenticators and acknowledgments. To limit the crypto overhead during bursts of updates, it also introduces a new composite message that allows multiple updates to be covered by a single authenticator (and thus by a single signature). We call this protocol variant *BGP with acknowledgments*, or BGP-A.

Log truncation: Routers require some storage for keeping the log. This storage does not have to be in the router itself – it could be on a separate blade, or on another computer – but capacity is limited, and log entries cannot be stored indefinitely. Therefore we allow routers to discard entries that are older than some time T_{max}, e.g., one year. Since the log contains periodic snapshots of the routing tables, discarding old entries does not destroy information about long-lived routes.

For routers to agree when T_{max} elapses, clocks must be loosely synchronized, e.g., within a few hours. NetReview enforces this by checking the timestamps on the authenticators. If a router's clock is not set properly, its messages will not be accepted by the adjacent routers.

If a log entry were not audited at least once during its lifetime, some faults could remain undetected. However, the typical audit period can be expected to be much shorter than the lifetime of log entries because ASes are likely to be interested in timely fault detection.

3.4 Auditing

To ensure that no fault goes undetected, the logs of each AS must be inspected regularly. Technically, it is possible to allow each AS to audit any other AS; however,

NetReview requires only that each AS audit the logs of its neighbors. Neighbors have a natural incentive to learn about each other's routing problems, and because of their existing business relationships, they are in a good position to take action if a problem is discovered. Also, recall our assumption that each AS has at least one diligent neighbor; this ensures that each log entry is properly inspected at least once.

To inspect an interval $I := [t_1, t_2]$ of a target's log, the auditor proceeds as follows:

1. If the auditor is not a neighbor of the target, it asks the target's neighbors for authenticators from interval I.
2. The auditor asks each of the target's border routers for a set of rules[2] and a signed segment of its log that covers interval I.
3. The auditor checks whether the following properties hold for the set of logs it has obtained:
   - **Consistency:** All authenticators match an entry in one of the logs.
   - **Conformance:** The sequence of messages in each log conforms to BGP-A.
   - **Compliance:** The target has followed each of the rules it has revealed.

3.5 Extracting evidence

When an auditor discovers an interval $I' := [t'_1, t'_2] \subseteq I$ for which one of the above properties does not hold, it extracts the corresponding log segment, starting at the most recent snapshot. Then it removes all entries that are not essential for checking the property (such as additional snapshots), as well as any parts of the first snapshot that are not needed to replay this particular segment. The result is a compact data structure that irrefutably ties the fault to the cryptographic key of the responsible AS, and thus (via the certificate) to its principal. This data can be used as evidence of the fault, and a third party can verify it independently without having to repeat the audit.

Once an auditor has obtained evidence, it notifies the local administrator, who can use the evidence in several ways. For example, if a best-practice rule has been violated, the auditor can choose to make the evidence publicly available; thus, it is possible to evaluate an ISP's performance by asking its neighbors for evidence of faults. If a private rule was broken, the evidence can be used to convince an arbitrator or a judge.

[2] In Section 3.9, we describe how the auditor can verify that the rules are genuine.

3.6 Consistency and conformance checks

The consistency check detects if the target AS has tampered with its log. Recall that each BGP-A message or acknowledgment contains a signed authenticator that is linked to a specific log entry, and thus to a specific point in the hash chain. If the target has returned a valid log segment, it will be consistent with all the authenticators; otherwise the log segment and the mismatched authenticator constitute a proof of misbehavior. Since neighbors collect each other's authenticators, and since we assume that each AS has at least one diligent neighbor, we know that any forged, omitted, or modified log entry is eventually detected by at least one neighbor.

The conformance check detects if the target has deviated from the BGP-A protocol. This is a purely syntactic check that does not consider *which* routes were announced, but rather *how* they were announced. For example, NetReview checks whether each message was well-formed and whether sessions were opened with the proper handshake before announcements were sent.

If the target AS passes the consistency and conformance checks, the auditor is convinced that the log accurately reflects the target's BGP traffic. The remaining check is designed to detect routing problems.

3.7 Extracting the routing state

The previous two checks are performed on logs from individual border routers of an AS. However, many routing problems arise because of inconsistencies between multiple routers. Therefore, the auditor must perform the compliance check based on the 'global' routing state of the AS, which it obtains by merging the logs from the individual routers.

NetReview models the 'global' routing state of an AS as follows. At any given point in time, the AS has a set of *peering points* with neighboring ASes, and for each peering point there are two routing information bases (RIBs): the *outRIB* contains routes that the AS has announced to its neighbor, and the *inRIB* contains routes that the neighbor has announced to the AS. Since BGP does not permit the announcement of multiple alternative routes, each RIB can contain at most one route for each prefix.

To determine how the target's routing state evolved over time, the auditor starts by loading the oldest checkpoint from each log, which contains a snapshot of the RIBs. Then it repeatedly picks the unprocessed message entry with the earliest timestamp across all logs, and it applies the updates in the message to the corresponding pair of RIBs. Thus, it obtains a sequence of routing states $S(t_i)$, where t_i indicates the time of the message that triggered the change. Note that each $S(t_i)$ contains a

pair of RIBs for each router or peering point; it is *not* a 'global' RIB for the entire AS.

3.8 Compliance check

The compliance check detects if the target has broken any of its rules. Conceptually, this is done by checking each rule against each of the routing states $S(t_i)$. Recall that even the complete set of rules does not necessarily amount to a full specification of the AS's routing policy; thus, checking rules is not equivalent to re-evaluating each routing decision the AS has made.

We have developed a simple specification language that ASes can use to formulate rules. In this language, a rule is written as a predicate on an individual routing state $S(t_i)$. For example, the rule

$$\forall c \forall r \in \text{outRIB}(18, c): \\ (\text{prefix}(r) \in P) \Rightarrow (123 \in \text{communities}(r))$$

stipulates that, when a route r belongs to a prefix from the set P and is announced to AS 18 over any peering point c, r must be tagged with the community 123. We give more details on the specification language and the rule checker in Section 4.

3.9 Rule commitment and access control

For the compliance check, the auditor must know which rules should hold for the target during the audited interval. Also, if a rule is violated, the auditor should obtain evidence that the rule existed at the time of the fault. The easiest way to accomplish both would be to simply record the rules in the tamper-evident log. However, since the logs are visible to each of the target's neighbors, this might reveal proprietary information about the target's routing policies.

Instead, we only require that ASes *commit* to their rules by logging a hash value $H(s_i, r_i)$ for each rule r_i. s_i is a 128-bit salt, which makes it difficult for an inquisitive auditor to learn sensitive information by checking for well-known rules, or to run a dictionary attack. On the other hand, if an auditor knows r_i and s_i a priori (perhaps from a peering agreement it shares with that AS, or because the AS has revealed them earlier), it can easily check whether the corresponding hash value is present. If not, it can use the log as evidence and file a complaint against the AS for breaking the contract.

Why would an AS commit to any rules at all, and why would it reveal a rule to an auditor? For example, ASes can use NetReview to enforce provisions from their peering contracts. The parties could agree to a set of rules and add them to their respective logs; they would then reveal these rules to each other, but not to anyone else. Or an AS could adopt a set of best-practice rules to highlight its good performance, and reveal these rules to everyone.

4 Writing and checking rules

NetReview includes a simple specification language that ASes can use to formulate rules. In this section, we describe this language in more detail, and we explain how rules in this language are evaluated.

4.1 Language design

The language includes three features we believe to be key for BGP fault detection. First, the language is *declarative* and refers to a high-level property, rather than to a specific algorithm for choosing routes. This makes rules easier to write and debug than, say, router configuration files. Moreover, many properties can be specified as rule templates that only require a few AS-specific parameters. A number of common templates are already included with NetReview.

Second, rules are *partial* specifications of the expected behavior. The above example only describes what should happen to routes that are announced to AS 18 and whose prefix is in P, but it does not say anything about the other routes. Thus, an AS can reveal a rule without revealing its entire routing policy. Also, we can vary the strength and number of rules and thus control how restrictive the checking should be.

Finally, rules are *time-local*, that is, they depend only on a small number of past and future states. This is possible because interdomain routing is essentially memoryless: whether or not a route is exported depends solely on which routes are *currently* available; it is irrelevant whether a route was available earlier, or will become available later.[3] This improves efficiency considerably, since NetReview only needs to remember a small number of routing states at any given time.

4.2 Specifying rules

Each NetReview rule consists of a set of constants and a set of predicates in first-order logic. The predicates are written using boolean operators, existential and universal quantifiers, and equality. They can use two functions called `inRIB(i,j)` and `outRIB(i,j)` to access the RIBs for a peering point j with a neighbor with AS number i. An optional third argument contains an interval operator.

[3] A notable exception is age-based tie breaking. We handle this by including the age of each route in the RIBs.

```
const setof(integer) asns = { 8, 9 };
forall cpref in affectedPrefixes, peer in asns {
  forall p1 in peeringPoints(peer), p2 in peeringPoints(peer) {
    (p1 != p2) => forall route in outRIB(peer, p1, intersect[now-5.0s,now]) {
      (prefix(route) == cpref) => exists route2 in outRIB(peer, p2, union[now-5.0s,now]) {
        (prefix(route2) == prefix(route)) and (sizeof(as_path(route2)) == sizeof(as_path(route)))
} } } };
```

Figure 2: Example rule in the syntax used by the NetReview rule checker. `intersect[a,b]` selects routes that were announced continuously between time `a` and time `b`, while `union[a,b]` selects routes that were announced at any point between time `a` and time `b`. `now` is the point in time for which the rule is evaluated.

Additionally, NetReview's rule checker has several built-in functions and operators for manipulating numbers, routes, sets, and sequences. These include basic arithmetic operators, functions for accessing the individual elements of a route, and set operators such as union, intersection, containment, and indexing. Figure 2 shows an example rule. This rule says that, when exporting a route to AS 8 or 9, the adopting AS must advertise AS_PATHs of the same length over all peering points with that AS.

4.3 Interval operators

Why do rules ever have to depend on future or past states? The reason is that, due to propagation delays and clock skew, RIBs from different routers may be slightly out of sync. Hence, there can be short intervals during which a route appears in an inRIB but in none of the outRIBs, or vice versa. To an auditor, this might look like a transient rule violation.[4]

To avoid false positives in this case, we must introduce a bit of leeway. NetReview's specification language contains two timing-related operators. Both operators take an interval $I = [t - \alpha, t + \beta]$ as an argument, where t is an instant in time and α and β specify how far the interval extends into the past and into the future, respectively. The *union operator* returns all routes that have been advertised at some point in I, and the *intersection operator* returns all routes that have been advertised continuously during I. This allows us to mask transient inconsistencies. For example, we might stipulate that a route may only be exported if a prefix of that route was available within two seconds of the current time, or that a route must be exported to some neighbor if it has been available for at least five seconds. We limit α and β to 60 seconds each; thus, the auditor must remember at most two minutes' worth of past or future states.

If a rule contains interval operators, it can miss actual transient faults that exist for less than $\alpha+\beta$ seconds. The interval needs to be no larger than the maximum propagation delay plus the maximum clock skew among the routers of an AS, so this is not a serious limitation.

[4]The use of a distributed snapshot algorithm such as [6] could avoid this problem, but it would require changes to the ISPs' internal route distribution mechanism.

4.4 Optimizations for checking rules

Conceptually, an auditor must evaluate each predicate whenever a) the routing state of the target AS changes due to an incoming or outgoing BGP-A message, or b) the value of an interval operator changes. For example, if a rule contains two intervals $I_1 = [t - 5, t + 3]$ and $I_2 = [t - 2, t + 3]$, the auditor must also evaluate each predicate five and two seconds *before* and three seconds *after* each routing change.

In practice, we can dramatically reduce the number of predicate evaluations using two simple optimizations. First, since rules typically consider each prefix individually, we can often restrict universal quantifiers to the set of prefixes that are actually affected by a routing change during the current evaluation. This set of prefixes is made available in a special variable called `affectedPrefixes`. Second, we can apply some simple query optimizations. For example, in the rule in Figure 2, NetReview combines the check for `prefix(route)==cpref` with the innermost `forall` quantifier, which reduces the quantifier to a simple projection.

4.5 Discussion

Even though our specification language is very simple, we have found that it is sufficient to describe many of the routing problems that have been reported in the literature, including origin misconfigurations [24], incorrect use of communities [24], incorrect extensions of imported routes [29], route deaggregation, redistribution attacks, and inconsistent path lengths [9]. We note that the particular details of the language are not critical to NetReview; NetReview just needs a way to specify and check constraints on the behavior of an AS. Our language could easily be extended or replaced without affecting the rest of NetReview.

No origin misconfiguration	$\forall a \forall p \forall r \in \text{outRIB}(a, p, \cap[t-40, t])$: ($\lvert\text{as\_path}(r)\rvert = 1 \wedge \text{prefix}(r) \in \text{ownPrefixes}$) $\vee$ ($\exists a' \exists p' \exists r' \in \text{inRIB}(a', p', \cup[t-40, t+5])$: $\text{prefix}(r) = \text{prefix}(r') \wedge \text{startsWith}(r, r') \wedge (\forall n \in r - r' : n \in \text{ownPrefixes})$)
Export customer routes	$\forall a \in \text{customers}\ \forall p \forall r \in \text{inRIB}(a, p)$: $((\forall n \in \text{as\_path}(r)$: $n = a) \Rightarrow \forall a' \in (\text{peers} \cup \text{providers})\ \forall p' \exists r' \in \text{outRIB}(a', p', \cup[t-15, t+15])$: $\text{prefix}(r) = \text{prefix}(r') \wedge \text{endsWith}(r, r'))$
Honor no-advertise community	$\forall a \forall p \forall r \in \text{inRIB}(a, p, \cap[t-5, t])$: NO_ADVERTISE $\in \text{communities}(r) \Rightarrow (\neg \exists a' \exists p' \exists r' \in \text{outRIB}(a', p')$: $\text{prefix}(r) = \text{prefix}(r') \wedge \text{getElement}(\text{as\_path}(r), 1) = a)$
Consistent path length	$\forall a \in (\text{customers} \cup \text{peers})\ \forall p \forall p'$: $(p = p') \vee (\forall r \in \text{outRIB}(a, p, \cap[t-5, t])\ \exists r' \in \text{outRIB}(a, p')$: $\text{prefix}(r) = \text{prefix}(r') \wedge \lvert\text{as\_path}(r)\rvert = \lvert\text{as\_path}(r')\rvert)$
Backup link	$\forall a \in \text{backups}\ \forall a' \in (\text{customers} \cup \text{peers})\ \forall p\ \forall r \in \text{outRIB}(a', p)$: $(\lvert\text{as\_path}(r)\rvert > 1 \wedge \text{getElement}(\text{as\_path}(r), 1) = a) \Rightarrow (\neg \exists a'' \in \text{providers}\ \exists p' \exists r \in \text{inRIB}(a'', p', \cap[t-5, t]))$

Table 1: Rules we checked in our experiments. Each rule is explained in Section 5.3. The variables a, a' are for AS numbers, p, p' are for peering points, and r, r' are for routes. $\text{inRIB}(a, p)$ and $\text{outRIB}(a, p)$ stand for the sets of routes imported and exported, respectively, to AS a over peering point p; they can be combined with an interval operator.[5]

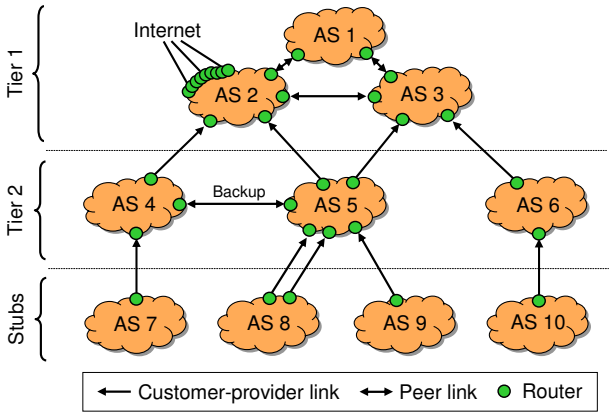

Figure 3: AS topology in our experiments. AS 2 receives updates from an Internet BGP trace.

5 Feasibility study

In this section, our goal is to demonstrate that NetReview (and, more generally, the fault detection approach) is practical. Using a prototype implementation of NetReview, we answer the following high-level questions:

- Are NetReview's rules expressive enough to describe common routing problems?
- How much storage and bandwidth is needed to maintain the tamper-evident logs?
- Is fault detection feasible at Internet scale?

5.1 Methodology

In NetReview, all communication related to a given AS occur among the direct neighbors of that AS. Hence, a small-scale deployment is sufficient to estimate the overhead. However, getting even a small number of contiguous Internet ASes to deploy experimental software would be extremely difficult. Instead, we used software routers to emulate a small AS topology in the lab, but we ensured that the routing table sizes and the amount of BGP traffic closely approximated those of real Internet ASes.

To achieve this, we injected an Internet BGP trace into one of the ASes, including a checkpoint of the initial routing table. From there, the routes were propagated to the other ASes via BGP, creating BGP traffic on each link and populating the other routing tables. This mimicked the conditions that would have occurred if our model topology had been part of the global Internet, so we could get realistic estimates for many performance metrics, e.g., how quickly the logs grow and how much time is required for checking. We found that, since the first trace already contained a route to each available prefix, injecting additional traces would not have increased the routing table sizes.

5.2 Experimental setup

Our NetReview prototype implements the basic system we have described so far, plus the additional techniques described later in Section 6, which enable NetReview to operate without a CA, in a partial deployment and with existing routers. These techniques add some overhead to our results, so the overhead of the basic algorithm would be lower than what we report here.

For our experiments, we set up a synthetic network of 35 Zebra BGP daemons [12], which form a topology of 10 ASes (Figure 3). Our network contains a mix of AS types, ranging from large tier-1 ASes to small stub ASes, as well as both customer/provider and peering relationships. This diversity allowed us to implement and check a variety of different routing policies. Note that AS 8 and AS 5 have two separate peering points, which will become important later.

[5]The interval sizes we use are worst-case values for a mirroring monitor (mainly due to MRAI timers). Much smaller intervals would suffice if the monitor is attached via port replicators or BMP [31].

For each AS, we configured a default routing policy that satisfies the Gao-Rexford conditions [11]. If a route is imported from a customer, it is exported to all neighbors; otherwise (if the route is from a peer or provider), it is exported only to customers. In some of our experiments, we vary this policy by injecting configuration errors or imposing additional constraints. Internally, each AS uses a full-mesh iBGP topology. We did not set up route reflectors because NetReview is oblivious to iBGP.

We injected routing updates from a RouteViews BGP trace [30] into AS 2. We used a 15-minute trace that was collected by a Zebra router at Equinix in Ashburn, VA, on January 27, 2008. The collecting router peers with nine other ASes. The trace contains 15,141 updates from these neighbors, and the corresponding RIB snapshot contains 243,198 unique prefixes. Thus, AS 2 behaved as if it were connected to the Internet in Ashburn, VA, and it exported a realistic set of prefixes to the other ASes.

NetReview's overhead depends in part on the number of neighbors an AS has. Unless otherwise noted, the numbers we report are for AS 5. Since 92% of Internet ASes have degree five or less [3], our results are representative of all but the largest Internet ISPs.

5.3 Rules we checked

In our experiments, we used NetReview to enforce five rules, which are shown in Table 1. In plain English, these rules state the following:

- **No origin misconfiguration:** An AS may only export a route if it owns the corresponding IP prefix, or if the exported route is an extension of another route that the AS is currently importing (motivated by [24]).
- **Export customer routes:** If an AS imports a direct route from one of its customers, it must export that route to its peers and providers.
- **Honor no-advertise community:** An AS must honor the NO_ADVERTISE community; it may not re-export a route that is tagged with this community.
- **Consistent path length:** When exporting a route to a customer or a peer, an AS must advertise AS_PATHs of the same length at all peering points (motivated by [9]).
- **Backup link:** An AS may only export a route via a backup path if its direct links become unavailable.

We chose these five rules because they can be used to detect real problems that have been reported in the Internet [9, 24, 29], and because they demonstrate the different types of conditions NetReview can verify (of course, each rule could be varied and customized in a number of ways). Note that the first two rules are very powerful; together, they can find almost all of the routing problems that were studied in [24]. In particular, the first rule covers AS_PATH manipulations, which are the main focus of secure routing systems like S-BGP (it actually goes beyond S-BGP in that it can also check for timely route withdrawal). The last three rules catch routing problems that would be difficult to find without a detection system, since they can only be detected by combining information from several routers and/or ASes.

5.4 Functionality check

We begin with a simple functionality check to show that the prototype is fully functional and works as expected. Recall that NetReview's design precludes false positives and false negatives if each AS is audited regularly.

We ran a series of six trials. In the first trial, we used the correct configuration for each AS. In the following five trials, we made a configuration change to a NetReview-enabled AS at some point during the experiment that caused one of the five rules to be violated. After each trial, we audited all the logs.

As expected, NetReview did not report any problems during the first trial. In each of the other trials, it reported the fault we had injected. The output also included the time interval in which the fault appeared, as well as the variable assignments (prefixes, AS numbers etc.) for which the corresponding rule did not hold. This is valuable for administrators because it shows not only where the fault occurred (in the audited AS) but also for *which* prefix the exported paths did not have the same length, *which* peering points were affected, etc.

5.5 Processing power

BGP-A speakers and monitors must generate and verify cryptographic signatures. The necessary processing time is a function of the number of messages they send and receive. In our experiment, the monitor in AS 5 sent 1,973 BGP-A messages and received 1,579 during the 15-minute period. Since all messages are acknowledged, this required 3,552 signatures to be generated and an equal number to be validated, on average four signatures and validations per second. On a 3 GHz Pentium 4, a 1024-bit RSA signature can be generated and verified in less than 3.5ms.

Unlike BGP messages, BGP-A messages can contain updates for multiple different routes, which explains why the number of messages is much lower than the number of routing changes in our BGP trace. This also limits the number of validations that are required when updates arrive in bursts. For example, if a router is restarted and re-

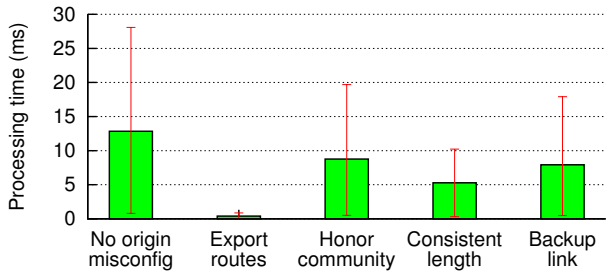

Figure 4: Average processing time required to check a rule over one second of log data (the error bars show the 5th and the 95th percentile). The speed is sufficient for checking multiple ASes in real time.

ceives full routing tables from its neighbors, it only needs to check one signature per routing table. This is in contrast to S-BGP [22], which needs to check a signature for every single route.

Auditors need processing power to extract the routing state from the logs, and to check it against the specified rules. In our experiments, we found that the processing time was dominated by rule checking, which in turn depends on the number of routing changes as well as the complexity of the rules. Our five rules can be evaluated independently for each prefix, so the first optimization from Section 4.4 can be used. It would take more time to check rules that depend on a large number of different prefixes, but we are not aware of any useful rules that have this property.

Figure 4 shows the average time required to check a one-second log segment against each of our five rules.[6] Our 15-minute log required 11,629 such checks, which took 41.5 seconds on a Pentium-4 workstation.

In practice, the checking time would also depend on the number and complexity of the rules the target AS is revealing to the auditor. There is little published information about the policies used by commercial ASes, so we cannot say how large a 'typical' set of rules would be. We already included a generic policy rule (rule #2) in our set, which may be sufficient for small ASes. Even if we assume that a typical set contains 20 rules (four times the size of our set), an AS with five neighbors would still only need a single workstation to perform real-time auditing. If an AS has more neighbors, it can spread the load across multiple machines, since rule checking can be trivially parallelized.

[6]The processing time varies considerably because some one-second intervals contain many updates, while others contain none at all.

5.6 Storage space

BGP-A speakers require storage for checkpoints, the tamper-evident log, and for the certificates that bind each key to the identity of an AS. An X.509 certificate with 1024-bit RSA keys is about 1kB. With web-of-trust signature chains (described in Section 6.1) and a typical AS-path length of four, each certificate is 5kB; thus, a database with certificates for 40,000 ASes would require approximately 195 MB.

The size of a checkpoint is dominated by the RIBs; it depends on the number of prefixes and peering points. One RIB with 244,000 prefixes and a 90-second history takes about 9.0 MB, so, if we conservatively assume that each prefix appears in every inRIB and every outRIB, a complete checkpoint for an AS with six peering points could take up to 108 MB. If the AS records one checkpoint every minute and keeps all checkpoints for one day, plus one checkpoint for each day of the last year, it would require up to 190 GB.

In our experiment, the log grew at a rate of about 332 kB per minute (without checkpoints). Hence, we estimate that one year's worth of log data would take about 166 GB. The log size is also a function of the number of peering points and the frequency of routing changes. Since the log mostly contains routing updates, its growth rate is roughly proportional to the amount of BGP traffic an AS generates. Recall that the numbers we report are for an AS with five neighbors; if an AS has more neighbors (and thus more peering points), its storage requirements are higher. For the largest ASes (UUNet has 2,652 neighbors), on the order of a hundred Terabytes of storage may be necessary to store the log for a year. However, the log would be distributed over thousands of routers.

Auditors require no permanent storage; however, it makes sense for them to cache a recent checkpoint for each AS they are auditing, so they do not have to download one repeatedly.

5.7 Message overhead

BGP-A speakers generate traffic for maintaining BGP-A sessions, for exchanging authenticators and for responding to audits. We look at each type of traffic in turn.

In terms of traffic, BGP sessions and BGP-A sessions are quite similar. If 1024-bit keys are used, a BGP-A message and its acknowledgment have 367 header bytes, while a BGP message only has 16. On the other hand, a BGP-A message can advertise many different routes, while a BGP message can only advertise one. In our experiment, AS 5 generated an average of 132 kB of BGP-A messages and acknowledgments per minute; these were equivalent to 135 kB of BGP messages.

Upon receiving a message or an acknowledgment, a BGP-A speaker detaches the authenticator and forwards it to the sender's neighbors. With 1024-bit keys, the size of an authenticator is 156 bytes; in our experiment, AS 5's neighbors sent 2.1 MB of AS 5's authenticators over the 15-minute period. However, authenticators are also collected from messages read during an audit, so the required traffic is *quadratic* in the number of neighbors: each neighbor audits each message and sends the corresponding authenticator to each of the other neighbors. This can be a problem for large ASes (e.g. UUNet). Therefore, authenticators from large ASes should be sent to only a subset of its neighbors. This does not affect NetReview's guarantees as long as the subsets used by all neighbors intersect in at least one diligent neighbor.

In our experiment, all audits were incremental; the auditor transferred a full checkpoint once and then retrieved only the log entries that were added since the last audit. In the limit, the required traffic is the size of the log times the number of auditors, plus some overhead for headers.

In total, AS 5 caused about 420 kbps of BGP-A traffic, including routing updates, auditing, and authenticators sent by the neighbors. This corresponds to the bandwidth of a typical DSL upstream, which is insignificant compared to the amount of traffic ISPs routinely handle.

5.8 Summary

Our experiments show that NetReview's simple rules are sufficient to describe common, nontrivial routing problems. Also, NetReview's resource requirements are moderate: in a typically-sized AS with five neighbors, routers must sign less than four messages per second on average, a single hard disk is sufficient to keep one year's worth of log data, and the total traffic is less than the capacity of a single broadband upstream link. Finally, we have demonstrated that fault detection is feasible at Internet update rates. By running the NetReview software on just a single workstation, an ISP can audit dozens of neighboring ASes in real time.

6 Practical challenges

In the previous two sections, we have shown that it is feasible to build a fault detection system with strong guarantees, and that its resource requirements are moderate. The goal of this section is to show how NetReview deals with the various practical problems that have hampered the deployment of previous solutions. In particular, we will show that NetReview can operate without a CA, that it can be effective in a partial deployment, that it can initially be deployed without upgrading any routers, and that it offers incentives for incremental deployment.

6.1 NetReview without a CA

Despite many proposals, deploying a global CA for prefixes and ASes has so far not found acceptance [19]. NetReview can use such a CA if it exists, but it does not require it. In the absence of a CA, we need to find replacements for two services that a CA provides: associating each key pair with a real-world identity, and certifying ownership of AS numbers and IP prefixes.

We solve this problem using a web-of-trust approach that is inspired by [33, 34]. Each AS initially generates a key pair and creates a self-signed certificate. Then it sends the certificate to its immediate neighbors, who append their own endorsement and forward it on to their neighbors, etc. The overhead for flooding certificates is not a concern, because the AS topology changes slowly.

Each AS obtains a database of all certificates, each with a chain of endorsements that corresponds to the shortest path between the local AS and the AS represented by a given certificate. Can these certificates be trusted? We can safely assume that each AS knows the true identity of the neighbor attached to each of its physical links. Moreover, we have assumed earlier that each AS has a diligent neighbor. This neighbor can detect if the AS signs a certificate that do not correspond to its true identity, or endorses a certificate that does not come from one of its neighbors. Thus, a node can (transitively) trust every certificate that is endorsed by one of its neighbors.

In addition, we require each AS to log a public pledge that specifies its current ASN and prefix ownerships.[7] ASes extract this pledge during audits and compare it to their database; if there is any change, they flood it to all other ASes. Thus, NetReview can detect if two ASes claim ownership of the same ASN or of overlapping prefixes, and it provides each with evidence of the other's claim. The conflict can then be resolved through existing mechanisms, e.g., by a mediator or a judge.

6.2 Partial deployment

It would be unrealistic to expect that all ASes adopt NetReview, much less that all ASes install the system at the same time. Therefore, NetReview must be able to work in a partial deployment, that is, it must be able to interact with non-participants via BGP.

By default, BGP-A speakers and monitors record only BGP-A messages in their logs, and auditors use only BGP-A messages to reconstruct the routing information. However, legacy neighbors have no components that speak BGP-A. If we simply omitted all routes imported from or exported to these neighbors, the information in the log might not be sufficient to evaluate many interest-

[7] Prefixes used for IP anycast [26] require special handling because they may be owned by multiple ASes simultaneously.

ing conditions. For example, if an AS acts as a provider for another AS, it may be required to export routes for all prefixes it knows about, even if the corresponding route is through a non-participant. Therefore, if an AS has legacy neighbors, its BGP-A speakers and monitors additionally record all the (unsigned) BGP messages they exchange with these neighbors.

Why keep this information in the secure record if a faulty participant AS can simply record whatever it wants? There are three reasons. First, we can isolate non-malicious faults such as misconfigurations or hardware failures, where the faulty AS still records correct information. Second, even if an AS lies about the routes it is importing or exporting via BGP, it must lie *consistently* to avoid detection by the auditors. For example, if the AS claims to have imported a certain route via BGP, it must re-export that route to each participating neighbor if required by its peering agreement, and it cannot export different versions to different neighbors.

Third, logging BGP messages enables an intermediate level of participation in NetReview. If a non-participant AS a is a neighbor of a participant AS b, a can act as an auditor and compare b's log to the BGP messages b actually sent, without fully deploying fault detection itself. All a needs is the NetReview auditor software and a current snapshot of its own BGP tables. If a finds a discrepancy, it can investigate it by contacting the participant AS b. This option could encourage neighbors of participant ASes to 'try out' fault detection.

Partial deployment requires an addition to the web-of-trust technique in Section 6.1. As long as the deployment is contiguous in the AS graph (which is likely if tier-1 ASes join first), the technique works as described. When a second 'island' of participants arises, at least one member of each island must exchange cryptographic credentials out-of-band. These members are then considered NetReview neighbors (even though they do not share a physical link), and they forward certificates from their respective islands to ensure that each AS has a full set. To increase the chance that ASes in small islands have a diligent neighbor, they also collect authenticators for each other and periodically audit each other's log.

6.3 Using existing routers

Requiring ISPs to upgrade or replace their routers to deploy NetReview would present a significant hurdle. Therefore, it is useful to have an intermediate solution that works with existing, unmodified routers. Our solution is to run the NetReview software on ordinary workstations, which we call *monitors*. The monitors speak both BGP and BGP-A; they observe all BGP traffic incident to the AS's existing routers and maintain BGP-A sessions with any monitors (or native BGP-A speakers where available) in adjacent ASes. The monitors also maintain tamper-evident logs and perform all cryptographic operations. Thus, the existing routers need not be modified.

There are two ways to configure a monitor [29]. A *proxying* monitor interposes on all BGP connections of its local AS. When it receives a BGP message from a local border router, it sends an equivalent BGP-A message to the remote BGP-A speaker (or monitor) and vice versa. A *mirroring* monitor snoops on the existing control connections, e.g., using a port replicator, the BGP monitoring protocol [31], or additional BGP sessions. Whenever it sees an outgoing message on the legacy BGP connection, it sends a BGP-A message with the same information over a separate connection to the neighbor's BGP-A speaker or monitor.

Mirroring monitors are safer because the routers do not depend on them. If a monitor fails, the routers can still send or receive routing updates via BGP and normal operation is not affected. On the other hand, mirroring monitors allow inconsistencies between the updates sent via BGP and BGP-A. Consider a case where a misconfigured or faulty router advertises some route A to its monitor and a different route B to the adjacent AS. The monitor would record route A in the tamper-evident log, and the AS could not be held accountable for route B.

To address this case, mirroring monitors maintain a third RIB for each peering point, which we will call *inRIB-BGP*. The inRIB contains the routes advertised via BGP-A as before, while the inRIB-BGP contains the routes received over the monitor's BGP sessions. Normally the two are identical; the scenario described earlier would manifest itself as an inconsistency between inRIB and inRIB-BGP in two adjacent ASes. Thus, an inconsistency cannot go undetected; however, an auditor cannot decide whether an inconsistency between inRIB and inRIB-BGP is caused by the audited AS or by its neighbor, and therefore must suspect both. Because BGP neighbors have a business relationship, they can be expected to swiftly sort out a demonstrated inconsistency between their advertised routes.

6.4 Incentives for deployment

If fault detection is to be deployed incrementally in the current Internet, we need good arguments to persuade ISPs to adopt it. Here, we present two arguments we believe to be compelling: ISPs can use fault detection as a distinguishing feature to attract more customers, and they can use it for root-cause analysis in the *entire* Internet, even in non-participating ASes.

Market forces: The first adopters of NetReview are likely to be large ISPs, such as tier-1 and tier-2 ASes, who tend to adopt new routing technology and best prac-

tices early. As a result, their routing performance is often excellent. These ASes can demonstrate their excellent performance by offering fault detection as a value-added service to their customers and thus distinguish themselves from the competition.

Once fault detection is on the market, competitors are encouraged to measure up by offering the service themselves. Thus, small islands of participants emerge. At this point, when a fault is caused by a non-participant, the participants can handle any complaints by proving that they are not the cause, and by tracing the problem to a non-participant just outside the island's perimeter, who must then handle the complaint. This creates an incentive for ASes to be inside the perimeter, and thus causes the islands to expand and the gaps between them to shrink.

Note that this approach works for NetReview because, unlike secure routing protocols like S-BGP, it is effective even in a small deployment of just a few ASes.

Root-cause analysis: As an additional benefit, participant ASes can use the fault detection system to diagnose faults *even if the cause is in a non-participating AS*. Since non-participants do not sign messages, do not maintain tamper-evident logs, and do not reveal any rules, we cannot guarantee that the diagnosis will always be accurate, and we cannot detect certain types of faults, such as policy violations. However, even an approximate diagnosis enables the AS to respond more effectively to faults.

Since non-participants do not have tamper-evident logs, we cannot directly apply auditing to find faults. Instead, we can use the participants' logs as a giant BGP looking glass that provides information about BGP updates from many vantage points. There are several proposed systems that can use this data to diagnose faults [10, 13, 21, 32]. In fact, because NetReview records a history of past states, it provides even more information than existing systems need; this could be used to develop even more powerful systems.

6.5 Accuracy in a partial deployment

When NetReview is used for root-cause analysis in a partial deployment, it returns a *candidate set* – a set of ASes that could have caused the fault. The size of this set depends on the size of the NetReview deployment. To estimate this dependency, we ran a simulation based on CAIDA's Internet AS topology [3], assuming a scenario in which an AS suspects that a given route has been spoofed. With NetReview in place, we can audit the participant ASes on the path and thus localize the fault to either a) a participant AS, b) a segment of non-participants between two participants, or c) the path suffix after the last participant. Here, we will ignore the possibility that the participant ASes record incorrect information.

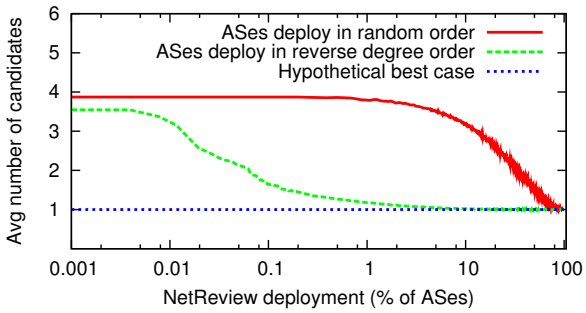

Figure 5: Fault localization under partial deployment. Shown is the average number of ASes that could have caused an observed failure.

For each deployment size, we simulated 10,000 trials as follows: we randomly picked an AS and calculated the shortest path to a random other AS using the Gao-Rexford conditions, then we picked a random AS on that path as the faulty AS and measured the number of candidates. We report averages over all 10,000 trials.

Figure 5 shows our results for two different deployment assumptions: either ASes deploy NetReview in random order, or in order of decreasing node degree. The first assumption is rather conservative; in practice, large ISPs typically run the latest routers, are among the first to apply best common practices and pride themselves on their good performance, so they are more likely to be early adopters.

In both cases, the average number of candidates starts at four (the average AS path length on the Internet). However, if the ASes with the most neighbors deploy NetReview first, the average decreases much more rapidly and reaches perfect localization with only a 15% deployment. The reason is that there are only about 12-15 tier-1 ASes; once these have deployed NetReview, faults can already be localized to one half of the path. 85% of the ASes in the Internet do not have customers of their own; once the other 15% participate, faults can be localized to one of them or to one of their customers.

This result shows that early adopters of a fault detection system like NetReview can derive considerable benefits from it; a deployment that includes the 0.1% highest-degree ASes would already be able to double the accuracy of its diagnoses. In contrast, fault prevention systems like S-BGP are only effective when they are already widely deployed.

7 Future work

In this section, we describe some advanced features that could be added to NetReview.

7.1 Simultaneously inspecting several ASes

NetReview inspects one log at a time, which is sufficient to detect protocol violations and policy violations. However, NetReview cannot detect problematic *interactions* between the policies of multiple ASes that way. An example is bad gadgets [14], which only arise when the routing policies of several ASes conflict in a circular fashion. To detect bad gadgets, NetReview would have to inspect the logs of multiple ASes simultaneously.

Technically, it is not difficult to fetch the logs from multiple ASes and to evaluate rules over multiple RIBs. However, routing policies are typically pair-wise confidential; thus, the check would have to be performed by a mutually trusted auditor. An alternative method to detect such policy conflicts, proposed in [15], is to have ASes annotate BGP advertisements with a *history* in a manner that preserves the privacy of the routing policies. Because NetReview records and publishes histories of BGP advertisements as part of its regular operation, this technique can be readily applied.

7.2 Detecting data-plane inconsistencies

In this paper, we have focused on providing fault detection for the *control plane* – the BGP announcements ASes send to each other. However, an AS could conceivably advertise one path in BGP and forward data packets on another, whether inadvertently or as part of an attack. NetReview already provides two mechanisms that can detect inconsistencies between the control and data planes: (i) it offers authoritative information about the route advertisement in the control plane, and (ii) it establishes the secure log that could also record observations about the data plane.

For example, suppose AS B advertises route "B C" to AS A but instead forwards A's traffic to AS D. If D passively monitors the traffic received from B, D can observe that A's packets are misrouted. D can add this observation to its log, and any auditors can thus obtain evidence of a data-plane inconsistency between B and D.

7.3 Internal audits

NetReview provides fault detection for BGP interdomain routing. It does not record any intra-domain routing messages in the tamper-evident log because these could reveal confidential information, such as the AS's internal topology.

However, NetReview could easily be adapted to cover intra-domain routing using a separate, private record. ASes could then perform internal audits to discover misconfigurations or compromised routers in their internal network, even when these routers have not (yet) caused a routing problem that would be visible to a neighbor.

8 Related Work

Detection: Anomaly detection techniques [7, 18, 21, 23, 36] use the BGP routing updates from one or more vantage points to build a *de facto* registry of the AS topology and prefix ownership. They raise an alarm upon receiving updates that disagree with the registry. Root-cause analysis (RCA) algorithms analyze BGP update messages from multiple vantage points to identify the AS(es) responsible for a routing change [4, 5, 10]. In RCA, each vantage point identifies a set of suspect ASes, then the sets are correlated to determine the potential culprit(s). The accuracy of RCA depends on the number and location of the vantage points. Unlike both RCA and anomaly detection, NetReview produces no false positives or false negatives, and it is not vulnerable to compromised ASes. In addition, NetReview can detect a larger class of faults, and it produces evidence that can be used to convince a third party.

AudIt [2] can determine which ASes are losing or delaying packets on the data plane. However, AudIt can only reveal the symptoms of a malfunctioning control plane, whereas control-plane fault detection can perform diagnosis.

Prevention: Secure routing protocols [20, 22, 33, 34] can ensure that (i) a route advertisement originates from the legitimate origin AS and that (ii) the AS-path of a route advertisement has not been modified or forged. On the one hand, secure routing protocols can prevent certain types of faults, whereas NetReview can only detect them; on the other hand, NetReview covers a larger class of faults, including policy violations (such as a faulty AS redistributing routes from one upstream provider to another), it can localize faults, and it provides incentives to avoid them. Perhaps more importantly, secure routing protocols do not provide appreciable benefits until many (if not all) ASes have adopted them, which explains in part why they have not yet been deployed, whereas NetReview is effective even in small deployments

N-BGP [29] uses trusted hardware to enforce a BGP safety specification for individual routers. Unlike N-BGP, NetReview does not require trusted hardware and it produces evidence of faults that can be verified by third parties. Moreover, NetReview is designed to check an entire AS's operation, not only against a safety specification but also against the AS's routing policy as specified in its peering agreements.

AIP [1] is a clean-slate redesign of IP that, among other things, would greatly simplify the deployment of a secure routing protocol. However, even if AIP were to replace IP entirely, it would be subject to the limitations of secure routing protocols described above.

Accountability: NetReview's tamper-evident log is based on the log in PeerReview [17], a general account-

ability framework for distributed systems. However, NetReview goes beyond PeerReview, which is based on assumptions that do not hold in interdomain routing. For example, PeerReview requires a certificate authority, it cannot operate in a partial deployment, it cannot protect the business secrets of ISPs, and it detects neither policy violations nor any other condition that involves more than one router.

9 Conclusion

In this paper, we have presented the design, implementation, and evaluation of NetReview, a fault detection system for interdomain routing. NetReview reliably detects incorrect behavior and links it to the responsible AS, while also enabling well-behaved ASes to prove they have adhered to the protocol and their routing policies. NetReview's correctness checks can detect and diagnose a wide variety of problems in BGP, including faulty equipment, buggy software, policy violations, and malicious attacks, which makes it an appealing alternative to specific solutions to any one of these problems. NetReview does not require changes to the underlying routers and is effective even in partial deployments. We believe that a fault detection system like NetReview can play an important role in improving the reliability, stability, and security of interdomain routing.

Acknowledgments

We are grateful to the anonymous reviewers and to our shepherd, Bruce Maggs, for their detailed and helpful comments. Paul Francis, Krishna Gummadi, Alex Fabrikant, Bryan Ford, and Sharon Goldberg provided valuable feedback on earlier versions of this paper. This work was done while Ioannis Avramopoulos was at Princeton University and supported by DHS grant W911NF-05-1-0417. Andreas Haeberlen was supported in part by US National Science Foundation grant ANI-0338856.

References

[1] D. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker. Accountable Internet protocol (AIP). In *Proceedings of SIGCOMM*, Aug 2008.

[2] K. Argyraki, P. Maniatis, O. Irzak, S. Ashish, and S. Shenker. Loss and delay accountability for the Internet. In *Proc. IEEE International Conference on Network Protocols*, Oct. 2007.

[3] AS relationships dataset from CAIDA. `http://www.caida.org/data/active/as-relationships/index.xml`.

[4] M. Caesar, L. Subramanian, and R. H. Katz. A case for an Internet health monitoring system. In *Proc. USENIX Workshop on Hot Topics in System Dependability*, Jun. 2005.

[5] J. Chandrashekar, Z.-L. Zhang, and H. Peterson. Fixing BGP, one AS at a time. In *Proc. ACM SIGCOMM Network Troubleshooting Workshop*, 2004.

[6] K. M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Trans. Comput. Syst.*, 3(1):63–75, 1985.

[7] Y.-J. Chi, R. Oliveira, and L. Zhang. Cyclops: the AS-level connectivity observatory. *SIGCOMM Comput. Commun. Rev.*, 38(5):5–16, 2008.

[8] N. Feamster and H. Balakrishnan. Detecting BGP configuration faults with static analysis. In *Proceedings of NSDI'05*, May 2005.

[9] N. Feamster, Z. M. Mao, and J. Rexford. BorderGuard: Detecting cold potatoes from peers. In *Proc. Internet Measurement Conference*, Oct 2004.

[10] A. Feldmann, O. Maennel, Z. Mao, A. Berger, and B. Maggs. Locating Internet routing instabilities. In *Proc. ACM SIGCOMM*, Aug.-Sept. 2004.

[11] L. Gao and J. Rexford. Stable Internet routing without global coordination. *IEEE/ACM Transactions on Networking*, 9(6):681–692, Dec 2001.

[12] GNU Zebra. `http://www.zebra.org/`.

[13] G. Goodell, W. Aiello, T. Griffin, J. Ioannidis, P. McDaniel, and A. Rubin. Working around BGP: An incremental approach to improving security and accuracy of interdomain routing. In *Proc. Network and Distributed Systems Security*, Feb 2003.

[14] T. G. Griffin, F. B. Shepherd, and G. Wilfong. The stable paths problem and interdomain routing. *IEEE/ACM Transactions on Networking*, 10(2):232–243, April 2002.

[15] T. G. Griffin and G. Wilfong. A safe path vector protocol. In *Proc. INFOCOM*, Mar. 2000.

[16] A. Haeberlen, P. Kuznetsov, and P. Druschel. The case for Byzantine fault detection. In *Proc. HotDep'06*. USENIX, Nov 2006.

[17] A. Haeberlen, P. Kuznetsov, and P. Druschel. PeerReview: Practical accountability for distributed systems. In *Proc. Symposium on Operating Systems Principles*, Oct 2007.

[18] X. Hu and Z. M. Mao. Accurate real-time identification of IP prefix hijacking. In *Proc. IEEE Symposium on Security and Privacy*, May 2007.

[19] Y.-C. Hu, D. McGrew, A. Perrig, B. Weis, and D. Wendlandt. (R)evolutionary bootstrapping of a global PKI for securing BGP. In *Proc. HotNets Workshop*, Nov 2006.

[20] Y.-C. Hu, A. Perrig, and M. Sirbu. SPV: Secure path vector routing for securing BGP. In *Proc. ACM SIGCOMM*, 2004.

[21] J. Karlin, S. Forrest, and J. Rexford. Autonomous security for autonomous systems. *Computer Networks, special issue on Complex Computer and Communication Networks*, Oct 2008.

[22] S. Kent, C. Lynn, and K. Seo. Secure border gateway protocol (S-BGP). *IEEE JSAC*, 18(4):582–592, Apr 2000.

[23] M. Lad, D. Massey, D. Pei, Y. Wu, B. Zhang, and L. Zhang. PHAS: A prefix hijack alert system. In *Proc. USENIX Security Symposium*, Aug. 2006.

[24] R. Mahajan, D. Wetherall, and T. Anderson. Understanding BGP misconfiguration. In *Proc. ACM SIGCOMM*, Sep 2002.

[25] O. Nordstroem and C. Dovrolis. Beware of BGP attacks. *ACM Computer Communications Review (CCR)*, Apr 2004.

[26] C. Partridge, T. Mendez, and W. Milliken. Host anycasting service. RFC 1546, Nov 1993.

[27] A. Ramachandran and N. Feamster. Understanding the network-level behavior of spammers. In *Proc. ACM SIGCOMM*, Sep 2006.

[28] Y. Rekhter, T. Li, and S. Hares. A border gateway protocol 4 (BGP-4). RFC 4271, Jan 2006.

[29] P. Reynolds, O. Kennedy, E. G. Sirer, and F. B. Schneider. Securing BGP using external security monitors. Technical Report TR-2006-2065, Cornell University, Computing and Information Science, Dec 2006.

[30] RouteViews project. http://www.routeviews.org/.

[31] J. Scudder, R. Fernando, and S. Stuart. BGP monitoring protocol, Nov 2008. Internet Draft, draft-ietf-grow-bmp-00.

[32] R. Teixeira and J. Rexford. A measurement framework for pinpointing routing changes. In *Proc. ACM SIGCOMM Network Troubleshooting Workshop*, Sep 2004.

[33] T. Wan, E. Kranakis, and P. C. van Oorschot. Pretty secure BGP (psBGP). In *Proc. Network and Distributed System Security Symposium*, Feb 2005.

[34] R. White. Securing BGP through secure origin BGP. *Internet Protocol Journal*, 6(3), 2003.

[35] J. Wu, Z. M. Mao, J. Rexford, and J. Wang. Finding a needle in a haystack: Pinpointing significant BGP routing changes in an IP network. In *Processings of NSDI'05*, May 2005.

[36] C. Zheng, L. Ji, D. Pei, J. Wang, and P. Francis. A light-weight distributed scheme for detecting IP prefix hijacks in real-time. In *Proc. ACM SIGCOMM*, Aug. 2007.

[37] E. Zmijewski. Longer is not always better. `http://www.renesys.com/blog/2009/02/longer-is-not-better.shtml`.