Concilium: Collaborative Diagnosis of Broken Overlay Routes

James W. Mickens and Brian D. Noble
EECS Department, University of Michigan
Ann Arbor, MI, 48109
jmickens,bnoble@umich.edu

Abstract

In a peer-to-peer overlay network, hosts cooperate to forward messages. When a message does not reach its final destination, there are two possible explanations. An intermediate overlay host may have dropped the message due to misconfiguration or malice. Alternatively, a bad link in the underlying IP network may have prevented an earnest, properly configured host from forwarding the data. In this paper, we describe how overlay peers can distinguish between the two situations and ascribe blame appropriately. We generate probabilistic notions of blame using distributed network tomography, fuzzy logic, and secure routing primitives. By comparing application-level drop rates with network characteristics inferred from tomography, we can estimate the likelihood that message loss is due to a misbehaving overlay host or a poor link in the underlying IP network. Since faulty nodes can submit inaccurate tomographic data to the collective, we also discuss mechanisms for detecting such misbehavior.

1 Introduction

Peer-to-peer systems scale because they distribute responsibility across many nodes. For example, in cooperative overlay networks, hosts can route messages to each other in a small number of hops using local forwarding state whose size is logarithmic in the total number of peers [17, 19]. When all peers behave properly, such designs lead to elegant, scalable systems. But what happens when some hosts misbehave? Real-life experience with large distributed services suggests that faulty local configurations inevitably arise [16]. Some nodes may also try to actively subvert the system. Thus, dependable peer-to-peer frameworks must expect that machines will occasionally drop messages, delete data, or otherwise misbehave.

Consider an overlay-level route that starts at host A, goes through B, and terminates at C. If C does not receive a message from A, did B shirk its forwarding responsibilities, or were there faulty IP-level links that prevented B from receiving the message or sending it to C? In this paper, we show how to ascribe blame in such situations, providing mechanisms for identifying fault points in end-to-end overlay routes. Once the system has detected a misbehaving overlay forwarder or a bad link in the core IP network, it can route around the problem. It can also notify the owner of the malfunctioning component, who may not be aware of the local fault. Both actions can improve the overall reliability of the distributed service.

Our new diagnostic system, named Concilium, generates probabilistic notions of blame using distributed network tomography [1, 10], fuzzy logic [5], and secure overlay routing [7]. By comparing application-level drop rates with network characteristics inferred from tomography, Concilium generates the likelihood that message loss is due to a misbehaving overlay host or a poor path in the underlying IP network. Unlike Fatih [15] or packet obituary systems [2], Concilium does not require modification to core Internet routers. In contrast to RON [1], Concilium detects hosts which contribute faulty tomographic data to their peers.

2 Secure Overlays

Structured peer-to-peer overlays provide a decentralized, self-managing routing infrastructure atop preexisting IP networks. Each host is associated with an overlay identifier. When a host must forward a message, it consults locally maintained routing state to determine the next hop. In overlays like Pastry [17] and Chord [19], the local routing state consists of two logical components. The *leaf table* points to the peers with the numerically closest identifiers to the local host's identifier. The *jump table* points to peers whose identifiers differ from the local one by increasing, exponentially spaced distances. Messages are typically forwarded using jump tables until the last hop.

Castro *et al* introduced *secure overlay routing* [7] to prevent malicious nodes from subverting the forwarding process. In a secure routing framework, messages are delivered with very high probability if the fraction of non-faulty hosts is at least 75%. Concilium uses several features of secure routing to protect its distributed tomographic protocol. We briefly describe these features before discussing Concilium in more depth.

Before a host can join a secure overlay, it must acquire a certificate from a central authority. The certificate binds the host's IP address to a public key and an overlay identifier. Since identifiers are static and randomly assigned, adversaries cannot deliberately move their hosts to advantageous regions of the identifier space. Hosts also enforce strict constraints on the peers which can occupy each jump table slot. For example, in standard Pastry [17], a peer in row i and column j of a routing table must share an i-character identifier prefix with the local host and have j as its $i+1$ character; there are no constraints on the remaining characters. In secure Pastry, the peer must be the online host whose identifier is closest to point p, where p is the local host identifier with the i-th character substituted with j. These stronger peering constraints, in concert with random identifier distribution, limit the fraction of malicious peers in local routing state to the fraction of malicious nodes in the total overlay.

Using a *density test*, a host can probabilistically detect when peers misreport their leaf sets. By comparing the average inter-identifier spacing in its own leaf set to that of a peer's leaf set, a host can identify advertised leaf sets that are too sparse. In the absence of such checks, an adversary could suppress knowledge of peers that it does not control, forcing routing traffic or data fetches to go through corrupt peers.

For performance reasons, peers maintain both secure routing tables and "standard" routing tables. Standard tables can use techniques like proximity affinity [8] to minimize routing latency or maintenance bandwidth; secure routing is only used when standard routing fails. Messages requiring Concilium's fault attribution must always be forwarded using secure routing. Other messages can be forwarded using either mechanism.

3 The Concilium Diagnostic Protocol

Concilium diagnoses faulty overlay routes using a multi-step process. First, hosts exchange their routing tables so that they can determine the first few hops that a locally forwarded message will take. Second, hosts test IP-level network conditions using locally-initiated network probes. By exchanging the results of these tests, individual peers synthesize a global picture of link quality throughout the Internet. By combining routing data with the collaborative map of network conditions, nodes can identify broken IP links and misbehaving overlay forwarders; the latter are defined as end-hosts which drop messages when the IP-level paths to their routing peers are good.

When a host is deemed faulty, Concilium issues a *fault accusation* against that host. Each accusation is provisional, since the accused host may be able to prove its innocence by showing that messages were actually being dropped further down the route. If the accused host can generate a verifiable *fault rebuttal*, Concilium will revise its original accusation. Otherwise, hosts may refuse to peer with the accused node or treat its behavior with extra suspicion.

In this section, we describe the Concilium protocol in the context of a particular implementation strategy. We then discuss alternative implementations.

3.1 Validating Routing State

To troubleshoot end-to-end overlay routes, Concilium must validate the routing state that peers self-report. Concilium validates leaf sets using Castro's test and introduces a new test to verify jump tables. Like Castro's leaf test, Concilium's jump table test is a density check. However, instead of examining the average inter-identifier spacing in a jump table, it checks how many slots are occupied. Jump tables with low occupancy are considered suspicious. For the sake of concreteness, we describe the test in the context of a secure Pastry overlay, but the test can be extended to other overlays in a straightforward manner.

In secure Pastry, overlay identifiers are ℓ characters long and each character can assume one of v different values. ℓ is typically 32 or 40, and v is usually 16. Each node maintains a jump table with ℓ rows and v columns. The identifier in row i and column j shares an i character prefix with the local host's identifier and has an $i+1$-th character of j. Assuming that identifiers are randomly distributed throughout the identifier space, the probability that a node does not have a particular prefix of length ℓ_{prefix} is $1-(1/v)^{\ell_{prefix}}$. The probability that an entry in row i of a routing table is filled is equal to one minus the probability that no identifier exists with the appropriate prefix. Thus,

$$Pr(entry\ filled\ in\ row\ i) = 1 - \left[1 - \left(\frac{1}{v}\right)^{i+1}\right]^{N-1} \quad (1)$$

where N is the total number of nodes in the overlay. Nodes can estimate N by inspecting the inter-identifier spacing in their leaf sets [13].

Let $p_{i,j}$ denote the probability that an entry in row i and column j is filled, as given by Equation 1. Each $p_{i,j}$ is an independent Bernoulli random variable, so the occupancy distribution for the entire table is governed by a Poisson binomial distribution. The mean and variance are

$$\mu = \frac{1}{\ell v}\sum_{i=1}^{\ell}\sum_{j=1}^{v} p_{i,j} \qquad \sigma^2 = \frac{1}{\ell v}\sum_{i=1}^{\ell}\sum_{j=1}^{v}(p_{i,j}-\mu)^2.$$

Computing exact values for the Poisson binomial distribution is intractable for non-trivial numbers of Bernoulli variables. Thus, it is difficult to directly calculate the likelihood that a table contains a particular number of occupied slots. Fortunately, since σ^2 is high, we can use a normal approximation with little loss in accuracy [12]. In this approximation, the mean μ_ϕ and the variance σ_ϕ^2 are

$$\mu_\phi = \ell v \mu \qquad \sigma_\phi^2 = \ell v \mu(1-\mu) - \ell v \sigma^2.$$

The cumulative distribution function for table occupancy is $\phi(\mu_\phi, \sigma_\phi)$ where $\phi()$ is the cdf for the normal distribution. To test whether an advertised jump table is too sparse, a host compares its local jump table density d_{local} to the advertised d_{peer}. If $\gamma d_{peer} < d_{local}$ for some small $\gamma > 1$, the peer's jump table is deemed invalid. In Section 4.1, we use $\phi(\mu_\phi, \sigma_\phi)$ to select γ based on the resulting likelihood of false positives and false negatives.

The occupancy test prevents malicious hosts from advertising jump tables that are too sparse. We also wish to prevent hosts from advertising tables that are too dense. Since identifiers are centrally issued, a misbehaving host cannot fabricate an identifier for an arbitrary jump table slot. However, a host can collect identifiers from peers that have gone offline and use these identifiers to inflate its advertised table density [7]. To protect against inflation attacks, Concilium requires a jump table entry referencing peer H to contain a signed timestamp from H. Whenever host G probes H for availability, H piggybacks a signed timestamp upon the probe response. Later, when G advertises its jump table, it includes the signed timestamps for each non-empty entry. Peers will reject the table if it has stale timestamps.

3.2 Collecting Tomographic Data

Each host H is connected to its routing peers by a set of links in the underlying IP network. These links induce a communication tree T_H whose root is H and whose leaves are H's routing peers. We define the forest F_H as the union of the tree rooted at H and the trees rooted at each of H's routing peers. Concilium's goal is to estimate link quality in F_H. To do so, each tree root periodically probes the link quality in its tree. Peers then exchange their tomographic results to create a collaborative estimate of link quality in F_H.

Before a host can initiate the tomography process, it must determine the physical IP links which comprise its tree. These link maps can be derived using tools such as RocketFuel [18]. Internet routes are often stable for at least a day [21], so topological data need not be fetched often.

Once the topology is known, hosts infer link quality using lightweight proactive probing and heavyweight reactive probing. Lightweight tomography uses the availability probes that hosts already send to their routing table peers [17, 19]. The period of these probes is a minute or less, and the duration of high loss events in IP links is on the order of tens of minutes [14]. Thus, H can use these preexisting probes to detect high intensity packet loss inside T_H. More specifically, H schedules a lightweight probe of T_H as a periodic task whose inter-arrival time is picked randomly and uniformly from the range $[0, max\_probe\_time]$; max_probe_time is on the order of one or two minutes. H probes its entire routing table at once using a simplified version of Duffield's striped unicast scheme [10]. H generates a single probe packet for each routing peer, but it issues these packets back to back. Since these packets will stay close to each other as they traverse shared interior routers, they emulate a single multicast packet sent to the leaves of a multicast tree. If H receives acknowledgments from all peers, it assumes that there is no link loss. Otherwise, it sends a few more probes to silent peers to determine if they are truly offline or situated along a lossy IP link.

If link loss is detected or H's application-level messages are not being acknowledged, H initiates heavyweight probing. Heavyweight tomography also uses striped unicast probing, but H sends many probes to each leaf using Duffield's full scheme. Loss rates for each root-leaf path are inferred using the number of acknowledgments received from each leaf host. Using maximum likelihood estimators, these end-to-end loss rates induce loss rates for each internal IP link.

When H initiates heavyweight probing, it asks its routing peers do the same. This ensures the availability of fine-grained, high quality tomographic data for the entire forest during the speculated fault period. To avoid probe-induced congestion, each peer waits for a small, randomly picked time before initiating heavyweight tomography.

After H has probed T_H using lightweight or heavyweight mechanisms, it sends a timestamped snapshot of T_H and its summarized probe results to its routing peers. The probe results for each path can be encoded in a few bits representing predefined loss rates. H signs the tomographic snapshot with its public key, both to prevent spoofing attacks and to prevent H from disavowing previously advertised probe results.

Each leaf node in T_H is one of H's routing peers, so H implicitly advertises its forwarding state when it publishes its tomographic data. This data also includes the signed freshness timestamps for each routing entry as described in Section 3.1. When a node receives a snapshot from H, it verifies all the signatures, checks the freshness of each entry, and performs the density checks. If any of these tests fail, the node may issue a fault accusation against H as described in Section 3.4. Regardless, the node archives H's snapshot. As the node receives snapshots from other peers, it constructs a distributed view of the forwarding paths emanating from its routing peers and the quality of IP links in these paths.

3.3 Error-checking Tomographic Data

Striped unicast tomography assumes that leaf nodes will return acknowledgments for received probes. A faulty or malicious leaf can try to respond to probes that were actually lost in the network, or drop acknowledgments for probes that were received. The former only affects inferences over the last mile to the misbehaving leaf, but the latter can ruin many inferences throughout the tree [3]. Fortunately, we can detect both types of misbehavior. To detect spurious responses to non-received probes, the probing node includes nonces in its probes. To detect leaves which faultily suppress acknowledgments, the probing node applies statistical tests to verify that the acknowledgment patterns of its leaves are consistent with each other [3]. Thus, an intentionally malicious leaf can accomplish the most damage by responding correctly to the

probes of other nodes, but misreporting the results of its own probes. We explore this issue further in Section 4.3.

We assume that interior IP routers can be faulty but not actively malicious. We assume that they do not interfere with probes or their responses in a byzantine way.

3.4 Attributing Fault

Armed with link measurements and routing information, each Concilium node can issue accusations for dropped messages. Suppose that at time t, host A sends a message to Z through B. By checking its copy of B's routing table, A can determine the host C to which B will forward the message. If A never receives a signed acknowledgment from Z, it checks its tomographic data for probes which test links in the path between B and C. If one or more links were probed as down, Concilium assigns blame to the network. Otherwise, Concilium determines that B was faulty. This judgment may be erroneous, since the true culprit may lie downstream from B. We describe how Concilium recovers from these mistakes in Section 3.5. For now, we restrict our attention to the original issuance of blame.

Let $B \rightarrow C$ represent the path between B and C, and let $probes$ be the set of probe results covering links in $B \rightarrow C$. We allow this set to contain results from probes initiated within the interval $[t - \Delta, t + \Delta]$, where Δ might equal sixty seconds. Let $probes(link)$ be the set of probes covering a particular link. For $p \in probes(link)$, let $p.l\_up \in \{0 \; or \; 1\}$ be the probed status of the link, with 1 representing a link that was up and 0 representing a failed link. Let $a \in [0, 1]$ be the accuracy of probes in diagnosing link failure. Returning to our running example, when A fails to receive an acknowledgment from Z, it ascribes blame to B as follows:

$$\begin{aligned} Pr(B \; faulty) &= Pr(B \rightarrow C \; good) \\ &= 1 - Pr(B \rightarrow C \; bad) \\ &= 1 - Pr(B \rightarrow C \; has \geq 1 \; bad \; link) \end{aligned} \tag{2}$$

where $Pr(B \rightarrow C \; has \geq 1 \; bad \; link)$ equals

$$max_{l \in B \rightarrow C} \left(\frac{\sum_{p \in probes(l)} [p.l\_up(1-a) + (1-p.l\_up)a]}{|probes(l)|} \right). \tag{3}$$

We use `max` as the `OR` operator from fuzzy logic [5]. In the context of Equation 3, it selects the link in $B \rightarrow C$ for which A has the highest confidence that it was bad, with each probe result weighed equally. For example, suppose that Q and R probe a link as down (0) and S probes the same link as up (1). If a equals 0.8, A believes that the link was bad with confidence (1/3)(0.8)+(1/3)(0.8)+(1/3)(0.2)=0.6.

Importantly, when A judges the trustworthiness of B, it does not incorporate B's probe results into Equation 3. This prevents a malicious B from influencing the amount of blame that A ascribes to it. For example, if A included B's probe results in Equation 3, B could reduce its level of blame by claiming that it probed a link in $B \rightarrow C$ as down.

Using Equation 2, A determines the amount of blame that it ascribes to B for a particular dropped message. If the blame is larger than a threshold described in Section 4.3, A assigns a *guilty verdict* to B; otherwise, A assigns a guilty verdict to the network. A maintains a sliding window of the last w verdicts that it issued for B, archiving the tomographic data used to make each verdict. If B receives m or more guilty verdicts in this window, A inserts a *formal fault accusation* into a DHT which exists atop the secure overlay. The insertion key for the accusation is B's public key, and the accusation contains all of the signed tomographic data that A used to derive its fault assessments. Insertions and fetches of the formal accusation are secured using Castro's techniques [7], and the statement is signed by A so that it can be held accountable for spurious accusations. When another host considers B as a routing peer, it first retrieves accusations against B from the DHT. For each accusation, the host uses the associated tomographic data to independently verify the fault calculations. If the host verifies the accusations, it considers B to be a "bad peer" and sanctions it according to network-specific policies.

3.5 Revising Incorrect Fault Attributions

As currently described, a Concilium node cannot ascribe blame beyond the next overlay hop. Returning to our running example, if A does not receive a signed acknowledgment from Z, and A estimates all links in $B \rightarrow C$ to be good, then A will always blame B for dropping the message, even if B successfully forwarded the message and it was actually dropped further downstream. To correctly ascribe blame in these situations, Concilium uses *recursive stewardship* of messages and *recursive revision* of fault accusations.

Whenever a peer along $A \rightarrow Z$ forwards a message, it treats the message as if it were generated locally—in other words, each forwarding peer expects to receive an acknowledgment from Z. If Z receives the message successfully, it routes its acknowledgment along the reverse forwarding path. If Z never receives the message or its acknowledgment is dropped along the reverse path, a chain of guilty verdicts will be issued. By considering them as a whole, Concilium can determine where blame should ultimately be placed. For example, suppose that D faultily drops A's message to Z along $A \rightarrow B \rightarrow C \rightarrow D \rightarrow ... \; Z$ and that all IP links are good. Using recursive stewardship, B and C will await an acknowledgment from Z. When this acknowledgment does not arrive, A will blame B, B will blame C, and C will blame D. D will not be able to blame a forwarding peer since it lacks incriminating tomographic data—D's peers in F_D will not have probed any links as down [1], and D cannot fabricate such probes itself because a node's own probes are ignored when calculating blame for that node. Thus, the accusation chain stops at D and nodes absolve themselves

[1] This assumes that the nodes in F_D are not colluding with D. We return to the issue of colluding nodes in Section 4.

of unfair blame by pushing locally generated verdicts upstream. First, C presents its guilty verdict against D to B. B examines the signed, timestamped tomographic data in the verdict, verifies the blame calculation, and amends its accusation against C to be an accusation against D. B presents its amended verdict to A. After A verifies the inference, it amends its accusation against B to an accusation against D. Innocent nodes have now been exonerated, and blame has been fairly attributed to D. Note that an amended accusation contains the signed, timestamped data from both the original verdict and the revision that was pushed upstream. This allows amended verdicts to be self-verifying.

Faulty nodes may not push revision information upstream. They do so at their own peril, since they will receive the blame for the message drop. For example, if C does not push its accusation against D to B, then B will not amend its original fault claim against C, and A will eventually blame C, not D, for the message drop.

A faulty node may receive a revision but refuse to update its local accusation. For example, A may receive B's blame against a node further downstream but continue to blame B. To guard against such misbehavior, B archives its local fault attributions and revisions. If another host believes that B is untrustworthy, it allows B to defend itself before any punitive steps are taken. The host presents B with the relevant formal accusations. If B can rebut these accusations using its local archives, the other host will recalculate B's trustworthiness in light of the new evidence.

3.6 Preventing Spurious Accusations

Up to this point, we have focused on detecting hosts which fail to forward messages. However, the original message sender can also misbehave. Suppose that each link in $B \rightarrow C$ is good. If A accuses B of dropping its message without actually sending one to B, other nodes will believe the accusation; they will verify the tomographic information in A's accusation and derive the same blame probability as A.

To prevent such spurious accusations, Concilium uses *forwarding commitments*. When A sends a message through B, B sends a signed statement to A indicating its willingness to forward the message. The commitment includes a timestamp, A's identifier, B's identifier, and the identifier of the ultimate destination Z. When A issues an accusation against B, it includes this forwarding commitment along with the relevant tomographic data and routing state. In this fashion, B can only be blamed for dropping messages that it agreed to forward. B can batch its commitments and asynchronously piggyback them upon its responses to A's availability probes. Like message stewardship and accusation revision, forwarding commitment is also recursive.

A malicious B may refuse to issue forwarding commitments for A's packets. Without support from core IP routers on the path between A and B, there is no way for Concilium to establish that A actually sent a message to B, or that B sent a forwarding commitment which A ignored. Lacking such knowledge, Concilium cannot comment on the trustworthiness of either peer. Fortunately, B's misbehavior can be detected by other mechanisms. For example, if overlay hosts are part of a decentralized reputation system such as Creedence [20], A can issue a vote of no confidence in B using this reputation system. Since honest hosts trust each other's votes, they will eventually determine that B makes a poor peer and treat it accordingly.

Note that standalone reputation systems cannot replace Concilium's full accusation protocol. Reputation systems allow a node to make a direct accusation against the next hop in a route, but they provide no structured way to propagate accusations against nodes that are farther downstream. Using recursive stewardship of messages and recursive revision of accusations, Concilium provides such a capability. Concilium also provides self-validating accusations which can be confirmed by arbitrary third parties.

3.7 Implementation Options

Up to now, we have assumed that each node performs its own tomographic probing. However, hosts which trust each other and reside in the same stub network can consolidate probing responsibility. For example, hosts could take turns issuing the probes for the multi-forest induced by their collective routing state. Alternatively, all hosts could defer probing responsibility to a shared administrative machine such as a RON gateway [1]. Either solution would make heavyweight probing less onerous, since the bandwidth cost for probing shared links could be amortized across multiple nodes.

As described in Section 3.4, a fault judgment is based on the acknowledgment of an individual message reception. If two peers exchange many packets, it may be useful for a single acknowledgment to cover multiple messages. The acknowledgment could indicate loss rates in several ways [15], e.g., through simple counters indicating how many packets arrived, or packet hashes identifying the specific packets which were received.

Concilium's goal is to find misbehaving overlay hosts and broken IP links, but it is agnostic about the response to its fault identifications. Broken IP links are often discovered quickly by the responsible ISP, so an overlay may simply avoid certain overlay paths until the fault is fixed [1]. With respect to faulty overlay hosts, Concilium allows each system to set an appropriate sanctioning policy. For example, accused hosts may not be trusted to forward sensitive messages.

If the overlay is used as a substrate for a higher level service such as a DHT, then honest nodes must not make local decisions to evict accused nodes from leaf sets. Otherwise, inconsistent routing [6] will arise and the higher level service may break. A network can mandate that a node be *universally* blacklisted if it receives accusations at a certain rate.

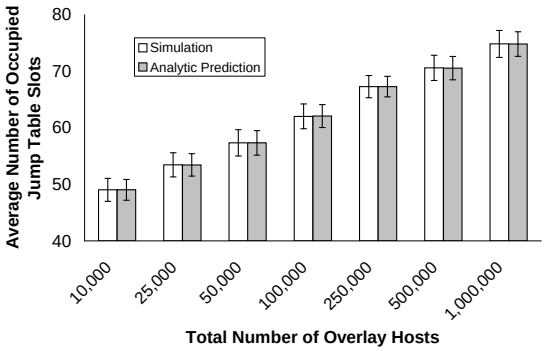


Figure 1. Modeling jump table occupancy

In such an environment, nodes would check the accusation repository before agreeing to peer with a new host. If the prospective peer was discovered to be faulty, it would not be added to the local routing table.

4 Evaluation

In this section, we use extensive simulations to evaluate the accuracy of our jump table check, the coverage properties of our collaborative tomography, and the error rate of our accusation algorithm. We also investigate the bandwidth overhead of the Concilium protocol.

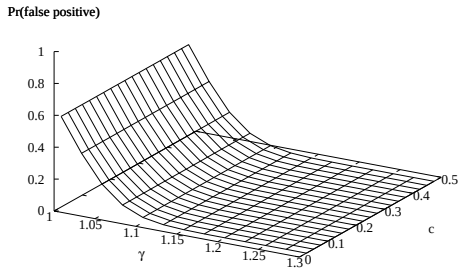
4.1 Jump Table Validation

Peers exchange their routing state so that Concilium can determine the IP-level tomographic data needed to make fault accusations at the overlay level. If peers can advertise incorrect routing tables without detection, innocent peers may be accused and faulty peers may go unpunished. Thus, the success of Concilium hinges on its ability to detect fraudulent routing advertisements. In this section, we analyze Concilium's jump table tests; we defer an analysis of leaf set checks to Castro's work [7].

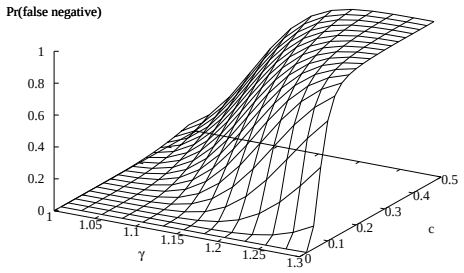
Our jump table test uses the cdf $\phi(\mu_\phi, \sigma_\phi)$ to model the distribution of occupancy fractions. Figure 1 compares the occupancy levels predicted by the analytic model with the occupancy levels seen in Monte Carlo simulations of table occupancy (y-bars indicate standard deviations). We see that the $\phi(\mu_\phi, \sigma_\phi)$ distribution accurately approximates real occupancy levels.

Our density test declares that a jump table is faulty if $\gamma d_{peer} < d_{local}$. The test can produce both false positives and false negatives. A false positive occurs if a non-faulty peer has a legitimately sparse jump table but is deemed faulty anyways. The likelihood of a false positive is equivalent to

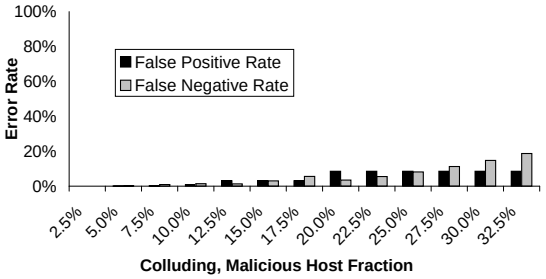
$$Pr(\gamma d_{peer} < d_{local}) = \sum_{0 \le d_i \le \ell v} \left[Pr(d_i) Pr(d < \frac{d_i}{\gamma}) \right]$$
$$= \sum_{0 \le d_i \le \ell v} \left[(\phi(d_i + \frac{1}{2}) - \phi(d_i - \frac{1}{2}))\phi(\frac{d_i}{\gamma}) \right]$$



(a) False positive probability.



(b) False negative probability.



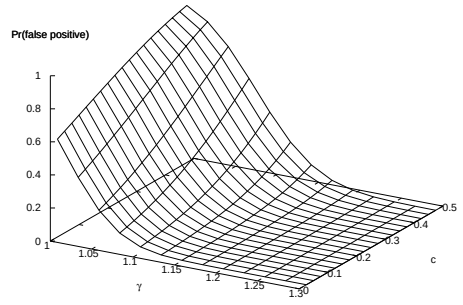
(c) Overall misclassification rate when γ is chosen to minimize the sum of the two error probabilities.

Figure 2. Error rates (no suppression attacks)

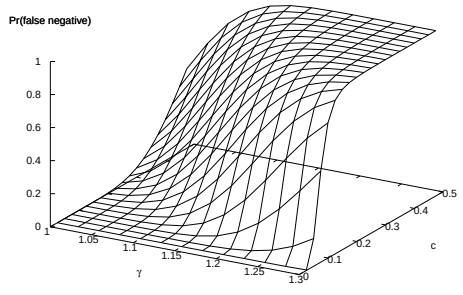
Figure 2(a) depicts the false positive rate as a function of γ and the fraction c of colluding malicious nodes. This graph assumes that malicious nodes may drop messages, but they may not try to go offline in a concerted attempt to skew local density estimates [7]. Thus, the false positive rate is independent of the fraction of malicious peers. Later in this section, we will revisit this graph in the context of suppression attacks.

The likelihood of a false negative is

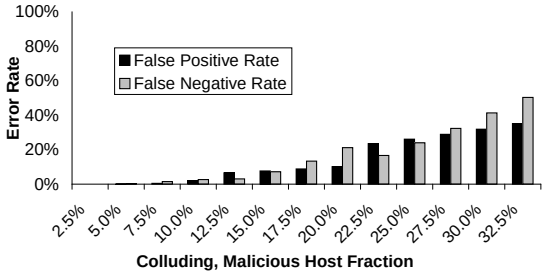
$$Pr(\gamma d_{peer} \ge d_{local}) = \sum_{0 \le d_i \le \ell v} [Pr(d_i) Pr(d < \gamma d_i)]$$
$$= \sum_{0 \le d_i \le \ell v} \left[(\phi(d_i + \frac{1}{2}) - \phi(d_i - \frac{1}{2}))\phi(\gamma d_i) \right]$$



(a) False positive probability.



(b) False negative probability.



(c) Overall misclassification rate when γ is chosen to minimize the sum of the two error probabilities.

Figure 3. Error rates (suppression attacks)

A false negative occurs when a peer advertises a jump table that only contains attacker-controlled nodes and the table passes the density test. Figure 2(b) shows the false negative probability in the absence of suppression attacks. Due to the properties of secure routing tables, an attacker is expected to control only c percent of all nodes in a jump table. Thus, the density of the attacker's fraudulent table is modeled as that of a legitimate table in an overlay with Nc total hosts. In the previous equation, when we calculate $Pr(d_i)$, i.e., the probability that the advertised jump table contains d_i nodes, we use Equation 1 but set the number of nodes to Nc.

Using Figures 2(a) and (b), we can choose the γ which minimizes some error metric. For example, Figure 2(c) shows the misclassification rate when γ is chosen to minimize the sum of the false positive probability and the false

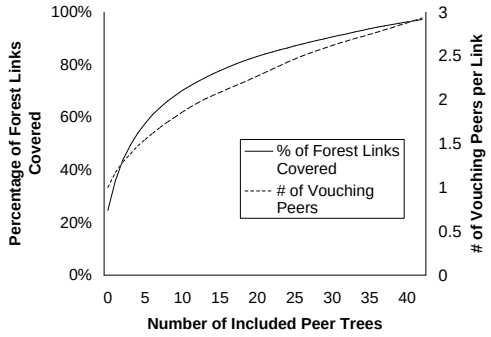


Figure 4. Trees Sampled vs. Forest Coverage

negative probability. If 30% of all peers are malicious and colluding, the false positive rate is 8.5% and the false negative rate is 14.8%. If 20% of hosts collude, the false negative rate decreases to 3.5%.

Figure 3 shows misclassification rates when adversaries can launch suppression attacks. We model these attacks by supplying our false positive/negative equations with the appropriately skewed versions of N as we did above. Like Castro's density tests for leaf sets [7], our jump table checks are not very reliable if more than 20% of hosts are malicious and colluding. For example, with a c of 20%, the false positive rate is 10.1% but the false negative rate is already 21.1%. Devising effective defenses against suppression attacks is an important area for future research. However, we note that c represents the largest set of *colluding* malicious nodes. The total number of malicious nodes may be much larger, but their power is limited by the extent to which they can coordinate the suppression of their identifiers.

4.2 Link Coverage

To test the coverage of tomographic probing and the accuracy of Concilium's accusation algorithm, we used a discrete event network simulator. The simulator modeled link failure, tomographic probing, the collaborative dissemination of probe results, and three types of message events (message sent, message acknowledged, message not acknowledged). The simulator placed a Pastry overlay atop an IP topology gathered by the SCAN project [11]. The topology contained peering information for 112,969 routers connected by 181,639 links. Following the methodology of Chen *et al* [9], we defined end hosts as routers with only one link and randomly selected 3% of these machines to be Pastry nodes. The resulting overlay possessed 1,131 nodes.

In the simulations, 5% of links were bad at any moment. Average link downtime was 15 minutes with a standard deviation of 7.5 minutes; this accords with empirical observations of high loss incidents lasting for a few tens of minutes [14]. Failures were biased towards links at the edge of the network [14]. To select a new link for failure, we randomly picked an overlay host and a random peer in that host's routing state. We then used a beta distribution with α=0.9 and

β=0.6 to select the depth of the link that would fail. Simulations lasted for two virtual hours. We did not model fluctuating machine availability since we wanted to focus on the fundamental properties of our fault inference algorithm.

Figure 4 shows the average percentage of IP links in F_H that are covered when H includes a given number of peer trees. If a node probes only its own tree, it can gather tomographic data for 25% of its forest links. Increasing the number of included peer trees results in large initial gains, but the improvement in coverage diminishes as more trees are included. This is because only a few trees are needed to cover highly shared links in the center of the Internet, but many trees are needed to cover all of the last-mile links that are only used by a few hosts.

As shown in Figure 4, gathering probe results from more peers increases the average number of hosts that test a given link and can potentially vouch for the status of that link at an arbitrary time. By increasing the number of vouching peers for a link, we improve the quality of tomographic inferences for that link. Greater link coverage also reduces the ability of malicious nodes to taint the diagnostic process by submitting bad tomographic data.

4.3 Accuracy of Fault Accusations

Accurate fault accusation requires accurate tomography. Duffield *et al* reported high levels of accuracy for striped unicast probing, with inferred link loss rates within 1% of the actual ones [10]. High accuracy rates have also been reported for other tomographic techniques [14]. In this section, we assume that hosts can identify whether a link was up or down with 90% accuracy.

Given the probe accuracy, we are interested in the amount of blame assigned to a forwarding peer when a message is dropped. Figure 5 depicts the probability distribution functions for the blame that Concilium assigns to faulty and non-faulty nodes. We generated the pdf by taking each triple of hosts (A, B, C) [2] and picking ten random times within the simulation period for A to route a message through $B \rightarrow C$. By comparing the actual link state along $B \rightarrow C$ to the tomographic information available to A at that time, we determined the amount of blame that A would assign to B if A did not receive an acknowledgment from the message recipient. B was a faulty node if it dropped a message despite $B \rightarrow C$ being good; it was non-faulty if at least one link in $B \rightarrow C$ was bad. Due to space constraints, we do not show results for the recursive revision of accusations; thus, the simulator ensured that a message was dropped either by B or a network link along $B \rightarrow C$, not by another peer or link further down the overlay route to the destination host.

Figure 5(a) depicts the blame pdf when all peers faithfully reported their probe results. Figure 5(b) depicts the blame pdf when 20% of peers colluded to maliciously flip their probe results. In the latter scenario, when a non-faulty node was being judged, malicious peers would always claim that their probed links were up (increasing the false positive rate); when a malicious peer was being judged, other malicious peers would always claim that their probed links were down (increasing the false negative rate). Comparing Figure 5(a) to Figure 5(b), we see that incorporating erroneous probe results into Equation 2 causes more blame to be assigned to non-faulty nodes and less blame to be assigned to faulty ones. However, Concilium can still make accurate fault accusations using a thresholding scheme which produces binary verdicts. For example, suppose that for any message drop, nodes receiving less than 40% blame are proclaimed innocent and all other nodes receive a guilty verdict. If all peers report their probe results faithfully, then innocent peers will receive guilty verdicts 1.8% of the time whereas faulty peers will receive guilty verdicts 93.8% of the time. If 20% of peers collude and contribute malicious probe results, then innocent peers will receive guilty verdicts 8.4% of the time and faulty peers will receive guilty verdicts 71.3% of the time.

A host issues a formal accusation against a peer if that peer accumulates at least m guilty verdicts for the w most recent message drops. To determine the false positive and false negative rates of formal accusations, let p_{good} be the probability that a non-faulty node receives a guilty verdict for a message drop, and p_{faulty} be the probability that a faulty node receives a guilty verdict; these probabilities are derived from the blame pdfs and thresholds as described in the previous paragraph. Let W be a random variable describing the number of guilty verdicts in a w-slot window. W is a binomial random variable, meaning that the error rates can be described as follows:

$$Pr(false\ positive) = Pr(W \geq m) = \sum_{k=m}^{w} \binom{w}{k} p_{good}^{k} (1 - p_{good})^{w-k}$$

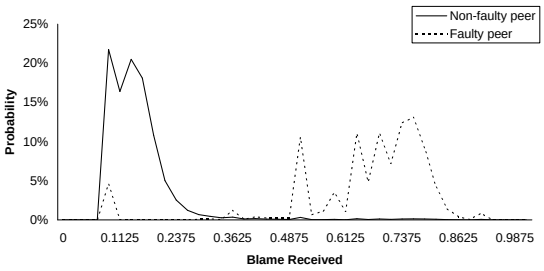
$$Pr(false\ negative) = Pr(W < m) = \sum_{k=0}^{m-1} \binom{w}{k} p_{faulty}^{k} (1 - p_{faulty})^{w-k}.$$

Figure 6 depicts the error rates with a blame pdf threshold of 40% and a sliding window size of 100. If all nodes faithfully report probe results, then we can drive both error rates below 1% with an m of 6. If 20% of hosts maliciously invert their probe results, we can achieve equivalent error rates with an m of 16.

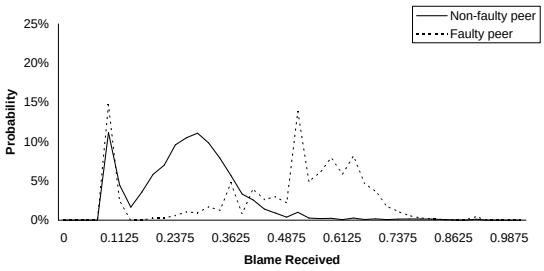
4.4 Bandwidth Requirements

Concilium has two primary sources of network overhead. Peers must exchange signed, timestamped copies of their routing state, and they must perform tomographic probing. We expect local routing state to reference μ_ϕ + 16 peers,

[2] This selection was constrained by the routing tables of each node, i.e., B had to be in A's routing table and C had to be in B's routing table.

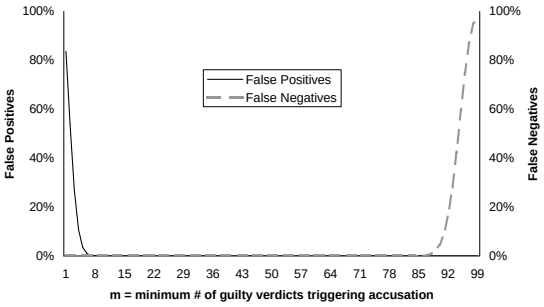


(a) The two pdfs are very distinct when nodes correctly report their tomographic data.

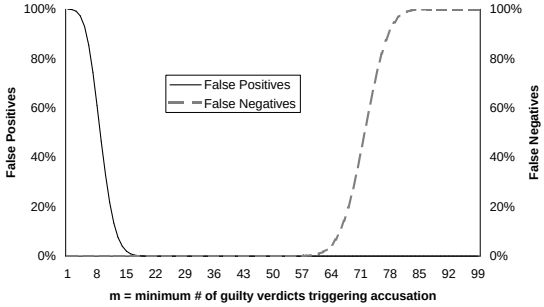


(b) The pdfs are less distinct when 20% of peers maliciously invert probe results, but Concilium can still make accurate judgments using a thresholding scheme.

Figure 5. PDFs for blame as generated by Equation 2 (max_probe_time=120 secs, Δ=60 secs)



(a) If nodes faithfully report probe results, an m of 6 drives both error rates below 1%.



(b) If 20% of hosts are colluding and malicious, an m of 16 drives both error rates below 1%.

Figure 6. Accusation error (w=100)

where 16 is the number of leaf nodes. Each routing entry contains a 16 byte node identifier and a 4 byte freshness timestamp. Using PSS-R [4] with 1024 bit public keys, both quantities plus a signature consume 144 bytes. The exchanged routing state also includes tomographic probe results for the IP path to each routing peer. As explained in Section 3.2, the results for each path can be encoded in a few bits. Assuming 1 byte for each path summary and a 100,000 node overlay, an entire advertised routing table is about 11.5 kilobytes. This overhead can be decreased by sending diffs for updated entries instead of entire tables.

In the absence of forwarding faults, lightweight tomography requires no additional bandwidth beyond that already required for availability probing. The outgoing bandwidth required for heavyweight striped probing of a tree is

$$\binom{|leaves \in T_H|}{2}(stripes\_per\_pair)(stripe\_size)(pkt\_size).$$

In a 100,000 node overlay, the average node has 77 entries in its local routing state. Suppose that each node sends 100 stripes to each ordered pair of peers, that each stripe contains two UDP probes, and that each probe is 30 bytes long (28 bytes for IP+UDP headers and 16 bits for a nonce). Probing an entire tree will require 16.7 MB of outgoing network traffic. Incoming probes will require no more than this amount and less if there are legitimately lossy network links.

The probing cost can be reduced in several ways. If IP multicast were widely deployed, we could reduce the probe traffic sent from the root of a tree to its leaf nodes. Also, as described in Section 3.7, cooperative hosts on the same stub network can share probe results, reducing the probing bandwidth for the collective.

5 Related Work

Packet obituary systems [2] allow end hosts to determine the autonomous system (AS) which dropped a particular packet. Each AS deploys an "accountability box" at each border link. When an incoming packet hits a box, the box records the next AS that the packet will traverse. Boxes periodically push these records along the reverse box paths, allowing each packet source to determine the last AS which successfully received their datagrams. Concilium differs from obituary systems in three ways. First, Concilium does not require the modification of core Internet routers. Second, Concilium protects and validates its network data using various cryptographic and statistical techniques. Finally, obituary systems cannot arbitrate between two adjacent ASes when the first claims that the second dropped its packet, and the second claims that the first never sent the packet. Concilium resolves such disputes using reputation systems.

Concilium assumes that end hosts may be malicious but core routers will not fail in a byzantine way. Fatih [15] is designed to detect core routers which maliciously drop or reorder packets. Each router maintains a summary of the traffic it has forwarded. Signed versions of these summaries are periodically exchanged with other routers, and misbehavior is detected by comparing summaries from routers that share links. Like obituary systems, Fatih requires modification to core Internet infrastructure.

In RON [1], each stub network has a special gateway which sits between the stub and the larger Internet. The RON gateways monitor the loss, latency, and throughput along the $O(N^2)$ paths which connect them. When a gateway must forward a locally generated packet outside its stub, it forwards the message through other RON gateways if the default IP path is poor. Like Concilium, RONs use active probing to detect link quality. The key difference is that RON always ascribes blame to the network—misbehaving RON nodes must be detected and removed by human operators. Concilium provides a mechanism for blaming the network *or* an overlay node.

6 Conclusions

In this paper, we introduce Concilium, a distributed diagnostic protocol for overlay networks. By aggregating peer-advertised routing state, Concilium determines forwarding paths at the overlay level. Using collaborative network tomography, Concilium discovers the IP links which comprise these paths and the quality of these links. By combining the topological and tomographic data with application-level message acknowledgments, Concilium judges whether dropped overlay messages are due to failures in the core Internet or failures in overlay forwarders. Concilium's fault accusations are self-verifying and robust to tampering, but they may place blame on nodes which are the victim of misbehavior further downstream in their routes. Thus, Concilium provides mechanisms to revise such incorrect accusations. It also has methods for detecting peers which publish faulty routing state or tomographic data.

References

[1] D. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proceedings of SOSP*, pages 131–145, Banff, Canada, October 2001.

[2] K. Argyraki, P. Maniatis, D. Cheriton, and S. Shenker. Providing Packet Obituaries. In *Proceedings of ACM SIGCOMM HotNets*, San Diego, CA, November 2004.

[3] V. Arya, T. Turletti, and C. Hoffmann. Feedback Verification for Trustworthy Tomography. In *Proceedings of IPS-MoMe*, Warsaw, Poland, March 2005.

[4] M. Bellare and P. Rogaway. The Exact Security of Digital Signatures – How to Sign with RSA and Rabin. *Advances in Cryptology–EUROCRYPT '96*, 1070:399–416, 1996.

[5] R. Bellman and M. Giertz. On the analytic formalism of the theory of fuzzy sets. *Information Sciences*, 5:149–156, 1973.

[6] M. Castro, M. Costa, and A. Rowstron. Performance and dependability of structured peer-to-peer overlays. In *Proceedings of DSN*, Florence, Italy, June 2004.

[7] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure routing for structured peer-to-peer overlay networks. In *Proceedings of OSDI*, pages 299–314, Boston, MA, December 2002.

[8] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron. Proximity neighbor selection in tree-based structured peer-to-peer overlays. Technical Report MSR-TR-2003-52, Microsoft Research, 2003.

[9] Y. Chen, D. Bindel, H. Song, and R. Katz. An Algebraic Approach to Practical and Scalable Overlay Network Monitoring. In *Proceedings of ACM SIGCOMM*, pages 55–66, Portland, OR, September 2004.

[10] N. Duffield, F. L. Presti, V. Paxson, and D. Towsley. Inferring Link Loss Using Striped Unicast Probes. In *Proceedings of IEEE INFOCOM*, pages 915–923, Anchorage, AK, April 2001.

[11] R. Govindan and H. Tangmunarunkit. Heuristics for Internet Map Discovery. In *Proceedings of IEEE INFOCOM*, pages 1371–1380, Tel Aviv, Israel, March 2000.

[12] R. Jurgelenaite, P. Lucas, and T. Heskes. Exploring the noisy threshold function in designing bayesian networks. In *Proceedings of SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 133–146, Cambridge, UK, December 2005.

[13] R. Mahajan, M. Castro, and A. Rowstron. Controlling the cost of reliability in peer-to-peer overlays. In *Proceedings of the 2nd IPTPS*, Berkeley, CA, February 2003.

[14] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. User-level Internet Path Diagnosis. In *Proceedings of SOSP*, pages 106–119, Lake George, NY, October 2003.

[15] A. Mizrak, Y.-C. Cheng, K. Marzullo, and S. Savage. Fatih: Detecting and Isolating Malicious Routers. In *Proceedings of DSN*, pages 538–547, Yokohama, Japan, June 2005.

[16] D. Oppenheimer, A. Ganapathi, and D. A. Patterson. Why do Internet services fail, and what can be done about it? In *Proceedings of USITS*, March 2003.

[17] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, November 2001.

[18] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with Rocketfuel. In *Proceedings of ACM SIGCOMM*, pages 133–145, Pittsburgh, PA, August 2002.

[19] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalabale peer-to-peer lookup service for Internet applications. In *Proceedings of ACM SIGCOMM*, pages 149–160, San Diego, CA, August 2001.

[20] K. Walsh and E. G. Sirer. Experience with an Object Reputation System for Peer-to-Peer Filesharing. In *Proceedings of NSDI*, pages 1–14, San Jose, CA, May 2006.

[21] Y. Zhang, V. Paxson, and S. Shenker. The Stationarity of Internet Path Properties: Routing, Loss, and Throughput. Technical Report, AT&T Center for Internet Research at ICSI, May 2000.