Providing Packet Obituaries

Katerina Argyraki[‡*], Petros Maniatis[†], David Cheriton[‡], Scott Shenker[*§]
[*]ICSI, [†]Intel Research Berkeley, [‡]Stanford University, [§]UC Berkeley

ABSTRACT

The Internet is transparent to success but opaque to failure. This veil of ignorance prevents ISPs from detecting failures by peering partners, and hosts from intelligently adapting their routes to adverse network conditions. To rectify this, we propose an accountability framework that would tell hosts where their packets have died. We describe a preliminary version of this framework and discuss its viability.

1. INTRODUCTION

The Internet is built around a best-effort service model that provides no *a priori* guarantees about when, or even if, packets will be delivered. Many have argued that this lack of guarantee played a key role in the Internet's success, as it enabled IP to run over a very wide range of network technologies using simple and scalable algorithms. While some clamor for adding quality-of-service assurances to the Internet, few if any believe that the best-effort model should be abandoned.

The Internet has also adopted the philosophy of not providing any *post hoc* information about the fate of packets. That is, the Internet provides neither assurances about what *will* happen to a packet nor information about what *did* happen to that packet. The rationale for not providing assurances – enabling flexibility of network technologies and simplicity of the forwarding path – does not apply to the lack of after-the-fact audits. The lack of accountability[1] more likely arises from strict adherence to layering and transparency according to which, from a host's perspective, all that matters is whether or not a packet was dropped, which can be determined by the endpoints themselves without help from the network.[2]

It has long been a central Internet tenet that applications should adapt to current network conditions, but often the notion of adaptation was limited to congestion control and application-level decisions (such as the coding scheme appropriate for the current bandwidth and loss conditions). However, a recent trend seeks to extend this to edge-controlled routing.[3] Proposals such as Platypus [8], NIRA [10], and WRAP [1] allow edge-systems to control the Autonomous System (AS) path of their packets. To make an informed decision about such paths when current service is poor, an edge-system needs to know at which AS(es) its packets are currently being dropped. Thus, providing some form of accountability should not be seen as putting the network in a more central role, but as helping return more control to the edges.

There is also a simpler and more venal rationale for providing accountability. Internet service is now a contractual business; end users pay their ISP, and ISPs have either customer/provider or peering arrangements with each other. Providing some form of accountability, in terms of which ASes are dropping packets, would help establish whether providers (and peers) are adequately performing their duty. Note that, while the previous rationale only requires feedback to the source, contractual issues suggest that ASes also would like to know the fate of the packets they forward.

The current Internet offers probing tools such as ping and traceroute that can help debug network problems. These tools are very effective in pinpointing long-lasting outages or persistent high-drop rates. However, because they reveal only whether probe packets are echoed, not what happened to previously sent packets, they fail in the face of interesting low-rate traffic patterns, e.g., malicious low-rate drop patterns, which exploit TCP's retransmission timeout to slow down flows well below their ideal rate [5]. As such, they have limited value when trying to make finely-tuned decisions about the reliability of a provider's service. In addition, one cannot expect that ISPs will remain so transparent to probing tools: what other industry allows free inspection of internal corporate infrastructure? We think that eventually edge-systems will not be given free probing access inside ISP networks.[4]

[1]*Accountability* is different from accounting. It is the property that enables activities on a system to be traced to specific entities, which may then be held responsible for their actions.

[2]For TCP congestion control and other purposes, the RTT of packets is also of interest, but this too can be determined by the endpoints.

[3]Edge-controlled routing dates back to the earliest source routing proposals but, after a lengthy quiescent period, there has been a recent upsurge of interest in this regard.

[4]Note that just leaving ICMP echo functionality available at border routers, though it retains traceroute and ping functionality at the AS granularity, still only provides information on the fate of the probe packets themselves, and thus is only useful to diagnose persistent failures.

Based on these considerations, we claim that the Internet would be well-served by an accountability framework that systematically provides information about the fate of packets. The purpose of this paper is to assess the feasibility of such a framework. We present an initial proof-of-concept design, guided by the following questions:

What information should be provided? While more information is always useful, the key information such a system should provide is where packets have been dropped, that is, *packet obituaries*.

To whom should this information be provided? Since both end-hosts and ASes (ISPs) have contractual expectations about the service given to their packets, a packet's obituary should be returned to every AS along the path of a packet as well as the source.

At what granularity should this information be provided? Ping and traceroute operate on a per-router granularity. But in terms of route choices and contractual obligations, the granularity in question is typically that of an AS. Moreover, providers may object to revealing more detailed information. Thus, we think that providing obituaries at the AS-level is sufficient.

Who will deploy the required infrastructure? We assume that ISPs will make accountability part of their commercial agreements with each other, and that soon thereafter customers will demand it of their ISPs. Thus, we expect that each ISP will deploy some accountability infrastructure that will communicate with its counterparts in the neighboring ISPs.

Should the mechanism be proactive or reactive? At first glance it might appear best to only provide obituaries upon request (reactive), since most hosts will not make decisions based on the received information. However, we think a proactive approach, in which packet audits are always sent at periodic intervals, is preferable for two reasons. First, ASes will need ongoing auditing of their peers, which requires proactive audits. Second, the accountability mechanisms should not make a bad situation worse when network conditions deteriorate. In the reactive model, many hosts will send requests only when the network is failing, thereby throwing gasoline on the networking fire.

Based on these five guidelines, we first develop a basic accountability framework (Section 2); inasmuch as this is a packet audit-trail design, it is loosely inspired from hash-based traceback [7] enhanced with proactive reporting. We discuss the technical challenges facing our design – viability, security and incremental deployment – in Section 3. We touch upon potential alternative designs in Section 4 and conclude in Section 5.

2. BASIC ACCOUNTABILITY FRAMEWORK

Infrastructure: We use *accountability boxes* (A-boxes) to implement the accountability framework. These A-boxes are deployed on the external links of each border router, either interposed between the router and the link, or positioned to passively tap the link. A-boxes are logically separate from routers, but the A-box functionality could eventually be integrated into routers.

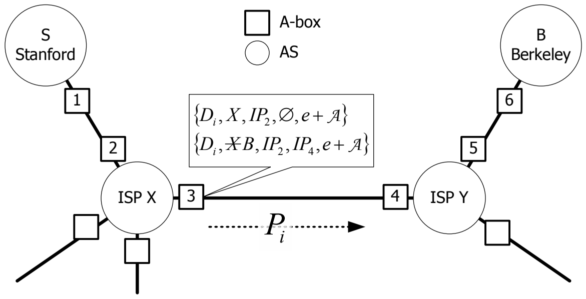
S
Stanford
A-box
AS
B
Berkeley
1
2
$\{D_i, X, IP_2, \varnothing, e+\mathcal{A}\}$
$\{D_i, \cancel{X} B, IP_2, IP_4, e+\mathcal{A}\}$
6
5
ISP X
3
4
ISP Y
P_i

Figure 1: A-box 3 forwards packet P_i and opens entry $\{D_i, X, IP_2, \emptyset, e + \mathcal{A}\}$. Later, it receives a feedback report with entry $\{D_i, B\}$, locates the entry with digest D_i and updates it.

Each AS operates its own A-boxes. These serve as its external accountability interface. AS internals are otherwise opaque to other ASes.

A-boxes produce, among other things, packet digests that uniquely identify packets. These digests need to have sufficient diversity to be unique among all packets traversing a high-speed link (e.g., OC-192) for an hour or so, and must be computable at wire speed. Sanchez et al. [6] have demonstrated hardware designs that justify this assumption.

Output of Framework: The accountability framework gives each AS feedback on every packet it originates or forwards. This feedback consists of the identity of the last AS that received the packet along its way to the destination. In Figure 1, Stanford sends packet P_i with digest D_i to Berkeley via ISPs X and Y. Stanford receives from X periodic reports whose entries have the form $\{digest, ASNum\}$. If packet P_i makes it through, the associated report entry is $\{D_i, B\}$, indicating that B received the packet. If, however, the packet is dropped within Y, then Stanford receives the report entry $\{D_i, Y\}$, indicating Y as the last AS to receive the packet. Hosts within an AS can access feedback reports for their packets via AS-specific mechanisms. For example, an AS can publish all received feedback reports in a central website, accessible by all of its hosts, or can implement the A-box functionality itself for incoming and/or outgoing traffic.

To provide this functionality, A-boxes remember the digests of packets they observe and periodically exchange feedback reports with other A-boxes. We now describe the resources and algorithms they use to do so.

Short-term State: For every observed packet, an A-box "opens" a new accountability entry in its state; each such entry has the following form:
$\{digest, lastASNum, prevABox, nextABox, expEpoch\}$
The value of the $digest$ field is computed over the packet's forwarding-invariant header fields and content; $lastASNum$ is initialized to the local AS number; $nextABox$ is initialized to $\emptyset$, the "uninitialized" value; $prevABox$ is set to the

IP address of the previous A-box traversed by the packet; and $expEpoch$ is set to the expiration time for this entry, where time is measured in *epochs*. We explain the duration of an epoch and describe how the values for $prevABox$ and $expEpoch$ are determined later in this section. For example, in Figure 1, when A-box 3 sees packet P_i during epoch e, it opens a new entry of the form $\{D_i, X, IP_2, \emptyset, e + \mathcal{A}\}$, where IP_j denotes the IP address of A-box j, and $\mathcal{A}$ is the maximum age of this entry measured in epochs.

Feedback Reports: Each A-box keeps an *epoch counter*, which it advances every T_r milliseconds, where T_r is the *epoch duration*. When the epoch changes, an A-box constructs a set of feedback reports: It removes some entries from its short-term state, groups them by their $prevABox$ values, chops off their last three fields, and sends each group to their $prevABox$. Removed entries are moved to the long-term state (see below).

Upon receipt of a feedback report, an A-box "merges" the contents of each reported entry with its local short-term state: For every reported entry, the A-box looks for a local entry with the same digest; if such an entry exists, the A-box replaces its $lastASNum$ with that included in the reported entry and updates its $nextABox$ with the IP address of the A-box that sent the feedback; if no such entry exists in the local state, the A-box ignores the reported entry. In Figure 1, when A-box 3 receives a report from A-box 4, it looks up a local entry with digest D_i, changes its $lastASNum$ from X to B, and sets its $nextABox$ to IP_4.

When a local entry is thus updated, we say it is "closed." During report construction for epoch e, an A-box only includes entries that are either closed, or have an $expEpoch$ field whose value is e.

Long-term State: This is a repository of feedback entries that the A-box has already reported. Such entries are retained for T_l hours, where T_l is a tunable parameter. The purpose of the long-term state is to allow an AS to answer follow-up questions on previous feedback reports. This is necessary when fraud is investigated, e.g., an AS misrepresents a report it received from a peer AS – more on this in Section 3.2.

Feedback Report Routing: Feedback on a particular packet must always flow backwards along the path traversed by that packet, A-box to A-box. In the current Internet, because of asymmetric routing, this "reverse A-box path" must be tracked and enforced by the A-boxes themselves. Therefore, each A-box remembers the last A-box traversed by a packet (in the $prevABox$ field) and forces the corresponding feedback through that A-box.

There are two types of edges in the reverse A-box path: *link* edges traverse a physical AS-to-AS link (e.g., from A-box 4 to 3 in Figure 1), and *intra-AS* edges traverse a single AS (e.g., from A-box 3 to 2). Link edges are simple to track: at link configuration time, the two A-boxes located at the edges of the link are given each other's address; as a packet exits the link, the second A-box records the address of the first A-box on the $prevABox$ field of the short-term entry for that packet. Intra-AS edges are more involved to track because the last A-box traversed by a packet within an AS has no way of knowing which was the first A-box in the same AS seen by that packet. This is akin to the problem of ingress point disambiguation tackled by Feldman et al. [4], albeit at the granularity (and speed) of packets instead of flows.

In our strawman design, we tackle this problem using packet encapsulation: The first A-box to see a packet coming into an AS creates a new IP packet with the same IP source and destination addresses as the original packet, and with its own IP address and the original packet as the payload. An A-box that sees a packet about to exit the AS decapsulates it and uses the A-box address in the payload to initialize its $prevABox$ field for the packet. An A-box does not encapsulate packets whose encapsulated length would surpass the intra-AS MTU; instead, the box sends within a *companion packet* – with the same source and destination as the original – the digest of the original packet and the address of the sending A-box. This companion packet is suppressed by the receiving A-box once its contents have been used to fill the appropriate packet entry.

This approach places A-boxes into the critical path, by requiring them to modify packets in flight. Although other, less invasive approaches exist, similar in philosophy to the companion packet approach above, they are far more complex and wasteful in processing and bandwidth. Fortunately, packet encapsulation is a popular technique, already present in the critical path of hardware routers. As such, it is a plausible design choice for A-boxes as well.

Short-term Entry Expiration: The maximum age $\mathcal{A}$ of a short-term entry depends on the AS distance of the A-box to the packet's destination. Consider packet P_i in Figure 1. If P_i is dropped in AS Y, A-box 3 receives feedback from A-box 4 on P_i in time $\mathcal{A}_4 T_r + RTT_{3-4}$, where $\mathcal{A}_j$ is the maximum age for P_i's entry in A-box j, RTT_{j-k} is the round-trip time between A-boxes j and k and T_r is the epoch duration. Unless $\mathcal{A}_4 < \mathcal{A}_3$, A-box 3's entry for P_i expires prematurely, and A-box 3 falsely reports that it was the last AS to receive P_i.

To compute the right $\mathcal{A}$ value per short-term entry, each A-box keeps a map of destination prefixes to AS-path lengths (which it can compute by periodically loading the BGP table from the closest border router). When a new entry is opened, the A-box looks up the AS-path length to the packet's destination, and sets $\mathcal{A}$ to the number of remaining A-boxes on the packet's path.

Our design is very sensitive to miscalculations of the short-term entry age. Unfortunately, the value computed from BGP tables may be inaccurate. Things become simpler if each AS has a "loss oracle" that tracks intra-AS packet losses; a loss oracle can be as simple as a per-packet ACK mechanism between two consecutive A-boxes. For example, if A-box 4 knows quickly that packet P_i has been lost in AS Y, it can close and report on P_i's entry immediately, without waiting for $\mathcal{A}_4 T_r$.

3. CHALLENGES

3.1 Viability

We first investigate whether this design imposes unreasonable burdens on the infrastructure. We consider a plausible hardware design, and outline its costs in terms of memory, bandwidth, and processing.

Memory: An A-box uses the following memory components: a Content Addressable Memory (CAM) module for the indexable fields of the short-term state; Static RAM (SRAM) for the remaining short-term state fields; an SRAM buffer for the feedback reports; a Ternary CAM (TCAM) module for the database that maps destination prefixes to AS-path lengths; and dynamic RAM (DRAM) for the long-term state.

Each short-term entry, in addition to the fields described in the previous section, contains a $closed$ bit, and a $clean$ bit. To give a rough estimate of the required memory, we assume the following field sizes: 80 bits for $digest$, 16 bits for $lastASNum$, 32 bits for each of $prevABox$ and $nextABox$, and 4 bits for $expEpoch$. The $digest$, $prevABox$, $expEpoch$, $closed$, and $clean$ fields are stored in the CAM, while the rest are stored in SRAM. Thus, each short-term entry occupies 118 CAM bits and 48 SRAM bits, for a total of 166 bits. A long-term entry is 2 bits shorter – no $closed$ nor $clean$ bits required.

The total amount of short-term memory M_{short} per A-box depends on the maximum packet rate C allowed on the link, the "batching period" $\mathcal{A}_{max}T_r$, and the short-term entry size E_{short}: $M_{short} = C \cdot \mathcal{A}_{max}T_r \cdot E_{short}$. The amount of long-term memory is similar, but depends on the long-term timeout and the long-term entry size: $M_{long} = C \cdot T_l \cdot E_{long}$. The buffer for the feedback reports is as large as the short-term memory – because the feedback reports built at the end of the same epoch take up at worst as much space as the entire short-term state. Finally, the prefix-to-length map depends on the number of Internet prefixes and the Internet diameter; currently, 1 MB is sufficient.

We consider an A-box located in an edge-network, for which $\mathcal{A}_{max} = 10$. We assume an average packet size of 400 bytes. For an epoch duration of $T_r = 100$ msec and long-term state period of $T_l = 1$ hour, an A-box on an OC-3 link (48.6 Kpps) requires 0.7 MB of CAM, 1.3 MB of SRAM, and 3.6 GB of long-term memory; an A-box on an OC-192 link (3 Mpps) requires 44.2 MB of CAM, 80.2 MB of SRAM, and 220 GB of long-term memory. Of course, with larger amounts of memory, we can support even higher packet rates; choosing what rate to support involves a trade-off between memory size and accuracy of our framework.

Bandwidth Overhead: For each observed packet, an A-box transmits one feedback entry to the previous A-box that forwarded the packet; each feedback entry is 96 bits wide (16 bits for $lastASNum$ and 80 bits for $digest$); assuming an average packet size of 400 bytes and excluding report headers, feedback reports introduce in each link bandwidth overhead equal to 3% of the link's throughput. Moreover, the first A-box of an AS that observes a packet encapsulates the packet adding its own IP address to the payload; this further increases bandwidth overhead on intra-AS links to 4.6% of the link's throughput.

Note that the exact bandwidth overhead is also dependent on the value of the maximum batching period $\mathcal{A}_{max}T_r$, which we have set to 1 sec above. For smaller batching periods, we may be able to represent reported packet digests with fewer bits than necessary for longer batching periods. In the hypothetical extreme of single-bit digests, the overhead is no less than 0.5%. Assuming that efficient compression methods for packet digests can be implemented for the batching design points in between, the bandwidth overhead will range from 0.5% to 4.6%.

Processing: An A-box performs three types of operations. First, it records a newly observed packet in short-term memory: it looks up a CAM entry with the $clean$ bit set and initializes it appropriately; to compute the maximum age, it looks up the AS-path length to the packet's destination in its TCAM map. Second, it handles a received feedback report: for every report entry, it looks up a CAM entry with the same digest and, if there, updates the entry. Third, it constructs a new report: it increments its epoch counter, looks up all CAM entries with set $closed$ bits or expired $expEpoch$ values, places them in a separate SRAM buffer and transmits them as feedback reports; then it stores the entire buffer into long-term RAM organized as a circular buffer. Long-term state can be handled similarly to how Snoeren et al. manage their historic memory [7] – responding to follow-up questions is not in the critical path, and can be handled by software on a best-effort basis.

Each A-box receives at most one feedback report entry for each packet it forwards. This means that, on average, the rate at which an A-box receives report entries equals the rate at which it records packets. Similarly, each A-box creates a report entry for each packet it forwards. Consequently, each A-box performs a total of two CAM lookups and updates plus a TCAM lookup for each packet it forwards. This is well within the capabilities of today's hardware.

3.2 Security Considerations

We have described our preliminary design for the case where ASes honestly report their feedback. Here we explore how our mechanism deals with malicious ASes. Given the criticality of carrier functionality, we assume that ASes have full control of their A-boxes, i.e., an honest AS always has honest A-boxes.

A malicious AS may exhibit at least two classes of behavior: First, it may claim that it missed a packet that it actually received. For example, in Figure 1, AS X receives packet P_i and drops it; then it claims that it never received P_i (by suppressing P_i from its feedback). Second, a malicious AS may misrepresent the feedback sent by downstream ASes. For example, in Figure 1, although AS X drops packet P_i, it blames the loss on Y (by forging a feedback entry that states Y as the last AS that received P_i).

We argue that an end-system can track the origin of a single feedback lie down to a pair of adjacent ASes. One way to do this is to allow an end-system to further investigate feedback on a certain packet by sending a post facto query, which is answered by all A-boxes on the path of the packet in question. Although feedback reports themselves are unprotected, post facto queries and their responses can be signed – most ASes already have SSL-protected web sites for customer service and incident tracking, so assuming they have a public signing key for this slow interaction path is not a stretch.

An end-system may wish to investigate either a suspect report or a random one (for spot-checking purposes). It routes a signed request for the packet's obituary to the original destination of the packet; each A-box that sees the request, responds with its signed long-term state entry for the packet or a signed statement that it never saw the packet. A single feedback lie will result in a single pair of A-boxes with conflicting responses; e.g, in Figure 1, AS X may respond "Y dropped the packet," while Y may respond "I never saw the packet."

If the end-system detects such a discrepancy, it sends the signed conflicting responses to each of the two involved A-boxes. This notifies them that either a communication failure occurred between them, or one of them lied. Although insufficient to convince a third party, this evidence at least convinces the neighbor of a misbehaving AS that its peer is faulty. This knowledge can inform further business decisions made by honest ASes with regards to evidently dishonest ones, both concerning end-system claims on individual reporting misbehaviors and for the future. We will prove that a single lie can be tracked back to a pair of adjacent ASes in a longer version of this paper.

3.3 Incremental Deployment

When not all ASes implement our scheme, feedback-enabled ASes must discover each other, so that their A-boxes can (logically) interconnect to exchange feedback reports. This discovery process is the subject of this section.

During discovery, each A-box on a link to an AS that does not support our scheme periodically sends announcement messages so that downstream A-boxes can discover it. A straightforward approach is to find one address prefix advertised by each potential destination AS, pick a random IP address within that prefix, and send to that address a "discovery request." The first A-box that observes this request absorbs it and sends back a "discovery response" with its AS number and IP address.

When adjacent A-boxes in a packet's path do not belong to the same or adjacent ASes, exposing a lying AS to its "neighbors" is less compelling; the "neighbor" of the liar may be many physical ASes away from the liar, which makes it difficult to take immediate action (e.g., interrupt business relationship). Our strawman design deals with feedback lies as described in Section 3.2, as long as the liar is one of a sequence of adjacent feedback-enabled ASes starting at the source AS.

3.4 Refinements

Here we tackle issues that our current strawman design leaves unresolved. For each issue, we either present how we propose to resolve it in subsequent designs, or why we find it inconsequential.

Packet Transformations: Our current design does not anticipate packet transformations such as fragmentation. We do not consider fragmentation an important issue; any packet source wishing to benefit from our framework can set the DontFragment flag – which is set in the majority of IP traffic already. Snoeren et al.'s design for packet transformations [7] may prove applicable to our mechanism as well.

Report Cascades: Long forwarding delays or errors in AS-distance estimation may cause a feedback report to reach an A-box after the related entry there has expired. In our current design, we drop such delayed reports; otherwise, pathologically timed delayed reports from downstream A-boxes could result in an A-box issuing up to $2d$ reports for the same entry (where d is the Internet diameter measured in number of ASes). An alternative would be to favor accuracy at the cost of occasional duplication of reports. In this approach, A-boxes can keep closed entries in short-term memory up to an additional age $\mathcal{B}$, as a *second-chance* cache. Updates to such closed entries cause the issuance of a report at the next reporting interval, until they expire at total age $(\mathcal{A} + \mathcal{B})T_r$. This increases our earlier short-term memory estimates by $(\mathcal{A} + \mathcal{B})/\mathcal{A}$, and our processing estimates by up to a factor of $2d$ at times of high delay.

Multicast: Extending this mechanism to multicast poses both mechanistic and conceptual challenges. The semantics of Internet multicast may require that the obituaries be sent to the receiver, not the source; the join message expresses the desire of the receiver, while the source has no knowledge of who should receive the packet. However, if there is a catastrophic failure that prevents any reports from getting through, then there is little utility in sending the reports back to the receiver. If instead reports are sent back to the source, they would have to be aggregated in some form to avoid implosion, and it is not clear what use the source could make of such reports. Thus, we are very unclear about how to treat multicast in this framework.

4. ALTERNATIVE DESIGNS

We developed our strawman design based on the five general guidelines presented in Section 1. Here we touch upon alternative guidelines and sketch the resulting designs.

Our basic framework provides per-packet obituaries. One could argue for less information: ASes care to know where losses occur, not necessarily where each individual packet got lost. Hence, one design alternative is to keep per-flow state instead of per-packet state: Each A-box opens a new short-term entry for every newly observed flow; the entry includes a packet counter, which is incremented with every observed packet that belongs to the flow. When the flow is over, the A-box closes the entry and puts it in a feedback report. A flow is defined based on IP source and destination addresses and packet interarrival times. Given recent evidence that Internet flows consist on average of 13 to 25 packets [2], such a flow-based approach reduces memory and bandwidth requirements by an order of magnitude, at the cost of providing coarser feedback.

Our basic framework returns a packet's obituary to every AS on that packet's path. One could argue for less functionality: When an AS drops a packet, inform only the packet's source (so that it routes its packets around the failure) and the culprit's peer (so that it detects if their service level agreement is violated). Hence, another design alternative is to break the mechanism into two parts, where one part carries feedback to the peer, and the other carries feedback to the source.

Sending feedback directly to the source ASes simplifies certain aspects of the accountability framework: First, feedback on a certain packet does not have to flow backwards along the path traversed by the packet; this removes the need for feedback report routing, packet encapsulation on intra-AS edges, and the discovery process described in Section 3.3. Second, each AS can send its feedback on a certain packet as soon as it observes the packet; this removes the need to accurately compute the AS-path length to the destination. Hence, this approach reduces processing requirements, at the cost of restricting the entities that receive a packet's obituary to the packet's source and the culprit's peer.

5. IMPACT ON THE INTERNET

The current Internet provides no explicit information about the fate of packets. While there are tools such as traceroute and ping to diagnose problems, they operate at the whim of ISPs (who will likely decide to block their probes at some point), and only provide information about the probe packets themselves, not about previously sent packets. To provide more detailed information, we proposed an accountability framework that delivers packet obituaries to each AS along the path, describing where packets were dropped (at the AS level). Our preliminary analysis suggests that this approach is viable, though there are several challenges that require further work.

We believe that an accountability framework would have a positive impact on the Internet, in several respects. Most importantly, the framework exposes ISP performance. Good ISPs will want to employ it, to prove to their customers that they are not responsible for packet losses. This may, in turn, drive the remaining ISPs to improve their (now measurable) service. In this sense, accountability could bring better ISP service by increasing competition on performance (which is now only dimly observable), not just on price (which is glaringly obvious).

The detailed performance information can also help hosts (or any edge-system) choose alternate routes to improve their performance. There are many proposals for letting hosts control their routes, but far fewer for how those hosts might gather the information necessary to intelligently choose their routes. By giving hosts the knowledge of which ASes are currently underperforming, they can narrow their search for better routes. This may even remove the need for Internet QoS mechanisms, since (ignoring access links) there are usually uncongested paths between two network points; to get good quality-of-service, edge-systems merely need to find those paths. Our framework, though not a complete solution to this problem, does provide useful information.

While this paper focused on the mechanistic details in order to establish viability, there is a deeper architectural issue at stake. The use of layering to hide implementation details from higher layers is a crucial aspect of the Internet architecture; correspondingly, edge-systems view the Internet as a black box, remaining ignorant of any network topology or structure. But equally crucial is the end-to-end principle of implementing as much functionality as possible in the edges. In particular, Internet applications should adapt to Internet conditions rather than expecting the network to adjust to their requirements. Without more knowledge of the Internet's behavior, the edge's ability to adapt is limited to congestion control and related behavior.

The accountability framework is an effort to provide structural information out-of-band. It does not violate layering, in that the semantics of IP are unchanged; the framework is an external vehicle for informing the host of network conditions. Many have called for an Internet knowledge [3] or information plane [9] that would expose network information to hosts. We view our accountability framework as a very primitive step in this direction.

6. REFERENCES

[1] K. Argyraki and D. R. Cheriton. Loose Source Routing as a Mechanism for Traffic Policies. In *Proc. SIGCOMM Workshop on Future Directions in Network Architecture*, Aug. 2004.

[2] A. Broido, Y. Hyun, R. Gao, and kc claffy. Diversity and Disparity in IP Traffic. `http://www.caida.org/outreach/papers/2004/diversity/diversity.pdf`.

[3] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski. A Knowledge Plane for the Internet. In *Proc. ACM SIGCOMM Conference*, Aug. 2003.

[4] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True. Deriving traffic demands for operational IP networks: methodology and experience. *IEEE/ACM Transactions on Networking*, 9(3):265–280, 2001.

[5] A. Kuzmanovic and E. W. Knightly. Low-Rate TCP-Targeted Denial of Service Attacks (The Shrew vs. the Mice and Elephants). In *Proc. ACM SIGCOMM Conference*, Aug. 2003.

[6] L. A. Sanchez, W. C. Milliken, A. C. Snoeren, F. Tchakountio, C. E. Jones, S. T. Kent, C. Partrige, and W. T. Strayer. Hardware Support for a Hash-Based IP Traceback. In *DARPA Information Survivability Conference and Exposition*, June 2001.

[7] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer. Hash-based IP Tracback. In *Proc. ACM SIGCOMM Conference*, Aug. 2001.

[8] A. C. Snoeren and B. Raghavan. Decoupling Policy from Mechanism in Internet Routing. In *Proc. Workshop on Hot Topics in Networking*, Nov. 2003.

[9] M. Wawrzoniak, L. Peterson, and T. Roscoe. Sophia: An Information Plane for Networked Systems. In *Proc. Workshop on Hot Topics in Networking*, Nov. 2003.

[10] X. Yang. NIRA: A New Internet Routing Architecture. In *Proc. SIGCOMM Workshop on Future Directions in Network Architecture*, Aug. 2003.