Building Accountability into the Future Internet

Jelena Mirkovic
Information Sciences Institute
University of Southern California
4676 Admiralty Way Ste 1001
Marina Del Rey, CA 90292
Email: sunshine@isi.edu

Peter Reiher
Computer Science Department
University of California Los Angeles
3564 Boelter Hall
Los Angeles, CA 90025
Email: reiher@cs.ucla.edu

***Abstract*—This paper proposes a future Internet architecture whose security foundations prevent today's major threats — IP spoofing, distributed denial-of-service attacks, distributed scanning and intrusions, and wide-spread worm infections.**

The core of the architecture are source signatures that are attached to each packet by its creator host. These lightweight, unforgeable signatures make senders accountable for traffic they originate. They also enable spoofing elimination close to sources since they are verified at each router hop.

The second layer of the architecture introduces route-independent, lightweight, unforgeable and short-lived packet tickets that act as capabilities. They indicate that the packet's destination agrees to receive traffic from a given source and eliminate some common denial-of-service attacks close to sources because they are verified at each router hop.

The top layer contains a reputation system that collects server reports about malicious client behaviors. Reports include verifiable proofs of malicious behavior, which prevents lying, and are aggregated into a client's reputation. Reputations provide information about previously unseen clients to servers that can use it to decide whether a client should be granted a ticket. Jointly, these three architectural layers introduce strong accountability into the future Internet.

I. Introduction and Motivation

Several large-scale security threats — IP spoofing, distributed denial-of-service (DDoS) attacks, vulnerability scanning, intrusions, and wide-spread worm infections — have long stayed unsolved because of the lack of *accountability* mechanisms in the current Internet. "Accountability" is defined as "an obligation to accept responsibility for one's actions", and it assumes that actions can accurately be linked to their sources and that sources can be punished for bad, and awarded for good behavior.

The current Internet is very far from this goal. IP spoofing makes it impossible to link packets to their sources. Even though recent research shows that spoofing is not very popular today [1], the mere *possibility* of spoofing prevents any punitive action against sources of malicious packets. This is the first nugget of wisdom we would like to emphasize: **accountability mandates perfect identification of actors**. Otherwise, any punishment/reward system can be tricked by attackers that assume good host identities to either bypass the system or to provoke punishment of good hosts. In addition to IP spoofing, the prevalence of DHCP, network address translation and mobile hosts makes IP addresses unreliable identifiers of traffic sources.

Even if destinations could reliably identify and drop spoofed traffic, many spoofed attacks would still consume Internet resources. For example, spoofed DDoS traffic could overwhelm the target's upstream link before its filter could drop it. This leads us to the second nugget of wisdom: **identification of sources must be cheap enough to be universal**. Thus routers could afford to verify the traffic source of each packet, and drop spoofed packets close to their sources. We propose lightweight source signatures that meet the perfect-and-cheap identification goals.

Assuming accurate and cheap source identification, the next question to solve on the road to accountability is how to exact rewards or punishments. The common approach is to punish malicious actors by filtering their traffic. Because unwanted traffic consumes Internet resources, **traffic filtering should occur as close to the sources as possible**, which is our third nugget. *Capabilities* were proposed in [2], [3] as a cheap way for a destination to communicate to routers on the traffic path which packets are desirable. A capability is an unforgeable, cheaply verifiable ticket that a sender obtains from the potential destination and attaches to its future packets. Routers verify tickets, drop packets with invalid tickets and rate limit ticket requests. Capabilities are short-lived so that a source which turns malicious could be easily cut off by refusing its future ticket requests. We build on current capabilities by proposing a novel, route-independent design.

While capabilities help eliminate unwanted traffic, destinations still need reliable means to decide if a source deserves a capability. The current design [2], [3] assumes that a destination always grants capabilities to new clients. After a short period of malicious activity, all attackers are identified and future capability requests are refused. This works fine for threats that inflict damage via ongoing traffic, such as flooding DDoS attacks, but not for threats that can succeed after a short interaction, such as intrusions and worm infections. The fourth nugget we emphasize is that **it is desirable that servers can identify malicious clients before having any interaction with them**. This is possible when clients have acted maliciously toward other servers and if there is an established, secure way of sharing this information, e.g., via client reputations. Several studies of scanners and botnets [4]–[6] have confirmed that hosts that were compromised in the past tend to misbehave in the future. A reputation system

that helps identify these repeat offenders can provide input to the capability-granting system and minimize the risk of zero-day threats. Such a reputation system is the third part of our proposed architecture.

A. Perfect Versus Operational Accountability

Our observation that accountability is crucial for unwanted traffic handling is not novel. Most recently it was made in the AIP publication [7]. However, our notion of accountability differs from AIP's in one key respect. We aim for *perfect accountability*, meaning that no entity can spoof another, and everyone can not only verify this but does so constantly for each packet. We achieve perfect accountability through novel anti-spoofing signatures that are cheap to generate and verify, yet unforgeable. AIP's authors aim for *operational accountability*, which leaves a small but present possibility of spoofing by well-positioned attackers. They achieve this goal by using expensive, public-key cryptography but utilizing routing information to minimize verification checks.

The main problem with operational accountability is that the mere possibility of spoofing enables attackers to trick any punishment/reward system based on observed behavior. This is the reason why the AIP architecture requires source cooperation to stop denial-of-service traffic, and employs packet caching at those sources to ensure that spoofed packets cannot lead to shutdown of legitimate sources. In our architecture, perfect accountability enables filtering of any malicious traffic at any point on the path, via capabilities. This is advantageous because wide-spread end-host cooperation is difficult to achieve. Another advantage we reap from perfect accountability is the possibility of proving without a doubt the origin of malicious traffic to a third party. This enables us to build client reputations and use them as input to capability-granting decisions.

B. Contributions and Overview

Our paper makes three contributions: (1) we propose a novel spoofing prevention scheme using lightweight, unforgeable signatures, (2) we propose a novel capability scheme built on top of unspoofable identities; our capabilities are to our knowledge the only ones that are route-independent, and (3) we propose a novel client reputation system, which cannot be tricked by lying participants. Such a system takes capability-based unwanted traffic handling to a new level: from mere mechanism to filter known unwanted traffic to a sophisticated framework that enables traffic prioritization based on its source's global behavior history.

Section II proposes lightweight, unforgeable, cheaply verifiable sender signatures that enable accurate linking of packets to sources. Section III proposes route-independent capabilities, and Section IV proposes an Internet-wide client reputation system, which facilitates recording and publicizing of repeat offenders. Servers file bad reports for clients that acted maliciously towards them. To prevent lying, servers also submit *proofs* – samples of malicious traffic with their reports. We discuss related work in Section V. We describe future work and conclude in Section VI.

II. Identity Spoofing Elimination

Our solution to identity spoofing attaches a source signature — a cryptographic signature dependent on the source identity and the packet's header and contents — to each packet. Routers are assumed to verify each packet's signature. We assume that the future Internet will separate naming and location services, and that packets will carry both identity (name) and location identifiers that will be globally unique.

Signatures must be *verifiable by anyone*, and *forgeable by no one*. These two properties are important for perfect accountability and they mandate using some method of asymmetric cryptography so that the secret needed to create the signature is only known to the identity owner. One mechanism that achieves this are public-key signatures, but they are too expensive to create and to verify at packet speeds.

A. Mechanism

We propose using *trapdoor hash functions with inversion property* for signing and verification. A trapdoor hash function is associated with the hash key (public key) *HK* and the trapdoor key (private key) *TK*. The trapdoor hash function $h(\cdot)$ is one-way hash function, which means that knowing *HK* it is cheap to compute $h(x)$ for any x, but it is very computationally expensive to find the input x that generates any $h(x)$. All one-way hash functions also have the collision-free property, which means that it is difficult to find two inputs x and x' so that $x \neq x'$ but $h(x) = h(x')$. Trapdoor hash functions have a property that finding collisions is easy if the trapdoor key *TK* is known. This property is used in an online/offline signature scheme proposed in [8] to sign a hash of a random message $\{m', r'\}$ offline, and later pair the signature with the original message m, $m \neq m'$ by using *TK* to find the value r so that $\{m', r'\}$ and $\{m, r\}$ form a collision, i.e., $h(m', r') = h(m, r)$. Thus offline signing can be slow (e.g., using public signatures) but the online signing requires only finding a collision and is fast. This makes online/offline signatures a good choice for signing streaming data such as network packets. However, the verification is still slow, so online/offline signatures cannot be used for the source verification service we propose.

Trapdoor hash functions with inversion property are hash functions such that finding the input x that generates hash value $h(x)$ is easy if the trapdoor key *TK* is known. We propose to use them in the following manner:

1) A source publishes the hash key *HK* and the verification token *V*. The source also enumerates packets it sends with an incrementing sequence number — each packet sent obtains the next sequence number even if the packet is retransmitted at the transport layer.
2) Verifiers store *HK* and *V* and keep a short record of sequence numbers recently seen from this source, e.g., by using a Bloom filter [9], to prevent replay attacks.

3) The source uses any hash function to compute a hash over the contents and immutable header fields, including the sequence number. This hash represents the message m. The source then uses the trapdoor key *TK* to find r so that $h(m, r) = V : SEQ_p$ where : denotes concatenation and SEQ_p is the sequence number of this packet. The packet's signature is r.
4) Verifiers check the packet's signature by calculating the hash over $\{m, r\}$ and verifying that it is equal to $V : SEQ_p$, and that the sequence number was not used previously.

The sequence numbers are needed to create diversity in hash values used for packet verification. The sequence number space should be reasonably large, so that it will take a long time (e.g., several days) to exhaust it. After the space is exhausted, the source must update V and potentially *HK* at verifiers, to prevent replay attacks.

B. Scalability and Cost

To make key management scalable, we propose a hierarchical signature scheme. Each host signs its packets using the proposed approach, and attaches its sequence number. As the packets leave the source autonomous system, the host-level signature is verified by a border router and replaced by the AS-level signature. The host sequence number is also replaced by the AS-level sequence number. Mobile hosts that change their location would inform the guest AS of their hash key *HK* and the verification token *V* during a registration procedure.

Entities in the Internet need only verify AS-level signatures to eliminate spoofing between ASes. In case of an untrusted AS that does not verify host signatures, hosts within this AS could still spoof one another. As a consequence, it is likely that all hosts within that AS would suffer collective punishment, as our capability and reputation mechanisms restrict their traffic. This is actually a desirable property, as it alerts the source AS to a problem that is in its power to fix.

Currently, routable ASes account for slightly more than half of the available 2^{16} space. While the AS number space is expected to increase to 2^{32} in the future, this increase must always be smaller than the increase in the size of routing tables. Routers of the future must be able to store and update their routing tables efficiently. Storage and update of $\{HK, V\}$ values will require much smaller overhead, because: (1) the number of routable ASes is always smaller than the number of routable prefixes, since some ASes announce multiple prefixes, (2) the size of $\{HK, V\}$ information is likely to be several hundred bits, while the size of a routing entry is often larger, (3) $\{HK, V\}$ values will be updated once every few days, while routing updates are far more frequent.

We assume that future Internet header will provide sufficient space for sequence numbers and for tickets described in the next section. According to calculations in [7], few Internet sources generate more than 50,000 packets per second. This means that the sequence number space of 32 bits could last close to three days. We estimate that the ticket size will be 6-7 times this, so total of 256 bits is needed to store identity and capability information. While this sounds large for today's IP header, it is much smaller than the header size proposed in [7]. Besides, security comes at a price — while our lightweight cryptography aims to keep processing cost low, we pay for it in key storage and IP header space.

The cost of the proposed signing is 5 modular exponentiations [8], while the verification cost is one hashing operation. This is acceptable because sources generate fewer packets than routers must forward; thus the signing cost can be slightly higher than verification cost. Publication [3] argues that a one-hash-per-packet operation is currently affordable for routers. We expect that future routers will be even better provisioned so the verification overhead should be acceptable.

The routers also need a Bloom filter to store source sequence numbers. To achieve a reasonable false positive rate the Bloom filter must be large enough to minimize risk of collisions. The false positive rate can be calculated as $p = (1-(1-\frac{1}{m})^{kn})^k \approx (1-e^{\frac{-kn}{m}})^k$, where m is size of filter in bits, k is the number of hash functions and n is the total number of key values that are in the filter [9]. For a 10^{-10} false positive rate, $k = 8$ and $m/n = 32$ we can remember $\approx 576,000$ 32-bit sequence numbers with a 32MB memory. With an average packet size of 400 bytes, this means that we can remember about 1 second of traffic on 2 GBps link.

C. Key Management

Since a node will periodically change its V and *HK* values, all the ASes must be updated with new values. We estimate that these updates will occur once per day via a push from the source to a representative node (server or router) in each AS. This node then updates all AS's routers with new values. The routers preserve old values for some limited time, to validate packets that may have been delayed in the network.

We can either use our lightweight signatures (with existing V and *HK* values) or traditional public-key approaches to validate new V and *HK* values. In case of scheduled, periodic updates, lightweight signatures that are already attached to packets carrying the update provide authentication without any additional cost. In case of key compromise and on the initial bootstrapping of the proposed anti-spoofing system, new V and *HK* values need to be authenticated using traditional public-key approaches. ASes would exchange traditional public keys using a chain-of-trust approach, as follows.

Peering ASes exchange their traditional public keys as they establish a peering relationship, using out-of-band communication. On bootstrap or key compromise, the AS creating a $\{V, HK\}$ update signs it with its traditional private key and disseminates the versions to all its AS peers. A peer already has the originator's public key needed to verify the authenticity. Upon successful verification and if the $\{V, HK\}$ values are new, it stores them and then modifies the message to propagate it further. Additionally, any unknown traditional public keys are extracted from the message, verified, and stored. Node modifies the message by adding its own public key to the list, and signing the entire message with its own traditional private key. It sends the message to all its neighbors

but the last sender-peer. The process repeats at each hop. The overhead of this communication is roughly one flood per several days per AS, and is affordable.

The authenticity of traditional private and public keys of ASes is proved in the established manner, via certificates issued by trusted certificate authorities (CAs). Thus, traditional public keys provided by neighbor ASes for other ASes are verified by the CA's signature, not merely the word of the neighbor AS. If a private key is compromised, we must revoke the certificate and issue new keys. These new keys are propagated invoking the mechanism described in the previous paragraph.

III. Reducing Unwanted Traffic

While capabilities have been proposed before [2], [3], these approaches generate route-dependent capabilities via cooperation of routers on the traffic path. Such capabilities become invalid if the route is changed or if there is multipath routing. Multipath routing is present even today [10] for load-balancing purposes, and it may be prevalent in the future Internet [11] for reliability reasons. Our proposed approach generates tickets (capabilities) at the destination, without cooperation from routers and without dependence on the route; it is thus well-suited for a dynamic routing environment.

We assume that all routers on the path verify the tickets in forwarded traffic and drop packets with invalid tickets. All packets in the Internet, including ticket requests and tickets, also carry the lightweight signatures described in Section II vouching for their authenticity.

A. Mechanism

The tickets are generated and used in the following manner:

1) The client issues a ticket request to a server. This request carries the "server ticket" [1], which the server will include in the reply so that its traffic is authorized to reach the client. The server tickets have identical structure and use to the client tickets, except that they have a different value in the type field. We explain the necessity of separating ticket types in the next section.
2) The server decides whether it will grant the access to the client, and on a positive decision generates the "client ticket" $T = \{sID, sAS, cID, type, lastValidTime, S_h\}$, where *sID* is the server's identity, *sAS* is the servers' AS number, *cID* is the client's identity, the ticket type is "client" and *lastValidTime* is the timestamp when the ticket should expire. $S_h = sign(sID, sAS, cID, type, lastValidTime)$, is constructed using the lightweight signatures proposed in Section II.
3) The server's border router verifies S_h and replaces it with the AS-level signature S_{AS}.
4) The client uses the ticket T in future packets, by attaching T and S_{AS} to each packet p, along with the value T_s, which is generated using the client's trapdoor hash function so that $h(T_s, m) = S_{AS} : SEQ_p$, where m is a hash over the contents and immutable header fields of p, including the sequence number SEQ_p.
5) Routers on the path verify tickets in all packets they forward. This is done by first verifying that the ticket is fresh. The freshness check assumes that the client's, server's and routers' clocks are loosely synchronized, which can be ensured by using the NTP protocol. A router then extracts *cID* and *sID* from the ticket and verifies that the source and destination identities in the packet match these fields. It extracts the S_{AS} and verifies that it is valid for the *sAS* and T. Finally, it hashes T_s and p with sender AS's *HK* and verifies that the output matches $S_{AS} : SEQ_p$.

When a client first requests a ticket from the server, the server has no direct experience of client behavior to verify that it is not malicious. The server may decide to obtain a client's reputation from the reputation system, described in the next section, and to grant tickets only to clients with a high reputation score, or it may decide to grant the ticket to each new client. Like in existing capability approaches [2], [3] tickets should be short-lived, since a client's behavior may change. We expect that the validity period of several seconds is short enough so that the server is not severely disabled if the ticket is misused for a flooding attack, and long enough to keep the ticket exchange cost affordable.

IV. Building Client Reputations

Reputation systems are commonly used to aid a user in selection of a trustworthy provider [12] [13] [14]. There is a significant body of research on provider reputation systems, with proven methods to handle the problem of lying participants and defamation attacks. We propose a *client* reputation system, which aids service providers in deciding to grant or decline tickets to a given client. During the ticket-granting process, each server may choose to rely more or less on a client's reputation score than on its personal, prior interaction with this client. This will likely depend on the server's prior experience with the client and the freshness of this experience versus the freshness of the reputation score.

A reputation can also be used during ticket-request floods to prioritize request handling. We assume that ticket requests are small, so they cannot overwhelm a server's network, but they can exhaust its CPU. A reputation check involves a table lookup and is cheaper than processing ticket requests. If a server's network is very limited, the server may request ticket request prioritization and filtering based on reputation scores from its upstream provider. The provider obtains reputation scores either from the server or from the reputation system.

A key insight behind client reputations is that a given host tends to be well-administered or poorly-administered over a considerable time, and that hosts that have behaved maliciously in the past warrant a lower trust since they are likely to misbehave in the future [4]–[6].

[1] The terminology used here to describe tickets is that a ticket type is identified by its user, not its issuer or destination. Thus, client tickets are used by clients to validate their packets to routers and servers, and server tickets are used by servers to validate their packets to routers and clients.

Client reputations differ from provider reputations in two aspects: (1) Providers tend to be consistently good or bad (e.g., an E-bay seller will consistently ship promptly) but clients tend to sporadically act maliciously towards a small set of targets. A compromised machine is not constantly misused for attacks – most of the time it is used by its owner for legitimate activities. Even when engaged in attacks, some attack types such as DDoS have a small target set. Thus existing provider-reputation approaches that detect lying through voting do not apply in case of client reputations: good votes would always outnumber bad ones. (2) Clients can afford to reject providers even at weak evidence of misbehavior, but providers that reject clients without a good cause suffer business loss. This makes the price of a false positive much higher in case of client reputations. Both these facts argue for (1) bad reports only, since good reports are expected to be plentiful even for malicious clients and (2) strong proofs of malicious behavior to minimize false positives.

A. Mechanism

The reputation system we propose works by collecting reports from servers about clients who have misbehaved. We assume servers have means of identifying misbehavior. The report contains the client's identity and the context of the misbehavior, i.e., "worm traffic with a rate of x scans to port y per second", "DDoS of type y with a rate of x packets per second", etc. We plan to closely specify the format for the context field in our future work, and we expect it will depend on the type of reported misbehavior. At the minimum, each context contains the packet rate and some description of malicious traffic features that can be verified by others if a traffic sample is provided. Server reports to a reputation center must be authenticated. We plan to provide this authentication with the anti-spoofing signatures from Section II that each packet carries already. After collecting reports, the reputation system serves them aggregated into a reputation score. Determining the correct aggregation method is part of our future work.

To prevent server lying, we require that each report be accompanied with a traffic sample proving that the alleged activity occurred. Using the proposed lightweight signatures and tickets, the reputation system can easily verify that the malicious traffic is authentic, recent, carries a *client* ticket (i.e. the behavior was unsolicited) and that its rate and features correspond to the context of the report. Since information needed to verify authenticity expires after a few days (when V and HK values are changed) reports must be filed soon after the malicious activity is detected. Note that the verification process does not prove that the traffic was malicious, it only verifies that it fits the reported context. Destinations are free to decide which traffic warrants a bad report based on private criteria.

We now explain the reason for separating server and client ticket types. Without this separation, a client could contact a server requesting a large file download, then use this traffic to prove that the server has sent it a packet flood. Separate client and server tickets prevent such an attack because the reputation system accepts only proofs that carry a client ticket.

Because a bad behavior may be rare, we must remember bad reports for a sufficiently long time to identify repeat offenders. On the other hand, a previously bad client that was cleaned and secured should have a way of redeeming itself and regaining a high reputation score. We plan to address these issues by providing short-term and long-term reputations. Short-term reputations are built by giving a higher weight to recent reports and discounting old ones, while long-term reputations are built using all reports submitted in a recent, long time interval. Short-term reputations are used by servers to accept redeemed clients' traffic during normal operation. Long-term reputations are used during an attack, which leads to dropping of redeemed clients' traffic; but this effect will be infrequent, short-lived and limited to those clients whose traffic reaches servers that experience attacks at the time.

A client that was an object of a bad report should be notified about this by the reputation system. The client should be provided with the complete bad report. In case that the client was compromised and the report was true, notification will alert the machine's owner about the compromise. Otherwise, if the client mistakenly contacted a malicious server, the notification will serve as a warning to avoid this server.

B. Deployment

For robustness and scalability reasons, as well as trust, the reputation system must be distributed, and a peer-to-peer design seems like a natural fit. A local reputation center could be deployed at each AS, or customer ASes could use the reputation center of their providers. This bounds the communication cost of the reputation center, because it collects and processes only local servers' reports. Reputation centers peer with each other to propagate reports or reputation scores. Messages between reputation centers, and reputation scores delivered to servers, are authenticated using anti-spoofing signatures already present in packets.

Compromised centers cannot forge false reports, but they can intentionally miscalculate reputation scores or selectively suppress true reports. Existing approaches from provider reputation systems can be applied to ensure that compromise of a reputation center cannot jeopardize credibility of client reputations in the entire system [12], [14]–[16]. For example, a center's peers can monitor its updates and vouch for correct score calculation. A server may need to contact several reputation centers for an update to minimize risk of lying.

Communication overhead for report submission may be large in case of large-scale security incidents such as a worm spread, so reputation centers may be overwhelmed. To control this overhead, we propose that a server aggregates all its reports within some interval into a combined report, and files it at the end of the interval. The combined report size may also be limited, e.g., by requesting that a server choose reports that describe the top N security incidents in the previous interval.

Reputation scores can be periodically (e.g., once a day) downloaded by reputation users (servers) from local centers.

These downloads can be staggered to minimize congestion. Since a client is presumed good in absence of bad reports, reputation users and centers need only store identities and scores of bad clients. Given that the largest botnet to date contained up to 50 million bots [17], such storage is affordable.

Reputation centers can also push reputations at times when numerous bad reports indicate a large-scale Internet incident such as worm propagation. To support this, local reputation centers would keep identities of servers associated with them on the notification list.

We now briefly discuss the bootstrapping process. When the reputation system is deployed, it will lack information about already compromised machines. As a result, it would initially regard all nodes as well-behaved, and would gradually develop information about which nodes are not. Certain aspects of our overall security architecture, such as the defense against identity spoofing, would work at once, since they are not based on knowledge of node behaviors. Others, such as issuing of tickets and reputation building, would improve over time. Given that misbehavior would quickly be detected and reported, the period before the architecture reaches its full effectiveness would be short.

V. Related Work

Unwanted traffic handling. Publication [3] proposed a DoS-limiting network architecture, TVA. The architecture enables routers to mark packets en route to the destination. If the destination is willing to accept this client's traffic, it returns the accumulated marks to the client as a capability. The routers are also engaged in filtering traffic with invalid capabilities. They prioritize traffic by giving priority to capability-carrying traffic, then to capability-request traffic and finally to legacy traffic. A main drawback of this architecture is that capabilities are route-dependent and thus become invalid during route changes or when multipath routing is used. Further, this architecture inflicts collateral damage on legitimate traffic during ticket-request floods, because all ticket requests are treated with equal priority, while our client reputations facilitate prioritization of requests based on clients' prior behaviors. TVA provides little guidance on how a destination could differentiate between good and bad clients, which is the major motivation for our client reputations.

In SIFF [2] authors propose a system very similar to TVA, with respect to how capabilities are generated and used. But SIFF capabilities are valid for a limited time, like our tickets, while TVA capabilities are valid for a limited number of packets. Recent work [18] attaches computational puzzles to ticket requests to prevent ticket-request flooding, whereas we use client reputations. Computational puzzles do not prove a client's "goodness" but just limit its request rate, and are biased against well-behaved but poorly provisioned clients. We have proposed reputation-based capabilities in [19]. These reputations are built locally at the server based on the recent source's traffic rate only, whereas reputations we propose here are built collaboratively to let servers benefit from experience of others.

In [20] authors propose PATRICIA, an edge network collaboration framework that enforces communication approval at the source and the destination network during attacks. Destinations express their approval via capabilities, and source networks can blacklist sources that misbehave or fail to obtain a capability. While the prevailing theme in [20] and our work is the same — traffic regulation via collaboration — the mechanisms are very different. Additionally, [20] does not address handling of misbehaving participants and does not elaborate on how to build malicious source blacklists, while we address both these problems via reputations.

Spoofing Elimination. Packet passports proposed in [9] are used to verify the source of a packet by attaching a sequence of marks, where each mark is created using a secret shared between the source and one AS on the path to the destination. This not only requires extensive secret sharing when deployment is large, but also cannot properly validate packets during route changes or multipath routing.

Spoofing prevention method (SPM) [21] associates a pair of source-destination ASes with a cryptographic secret exchanged between them, and carried in the packets. Packets are checked for the proper mark as they exit the source AS and on entry to the destination AS. SPM provides lower security than our solution because the packet mark is not bound to the specific packet, which allows sniffing. Even if this were fixed, use of a shared secret for authentication cannot provide the perfect accountability that the future Internet needs. Other spoofing prevention approaches [22]–[26] do not eliminate spoofing completely, but only provide means to routers [22]–[24] or to the target [25], [26] to detect and filter some amount of spoofed traffic.

Client Reputations. In [27] Allman et al. propose an architecture for behavioral history that could be applied to "actors" such as Internet hosts, mail servers, mail addresses, etc. This is similar to our client reputations, but the system is discussed at a very high level, which prevents direct comparison. One vulnerability of [27] is that lying is handled by valuing higher those reports that were labeled as useful by other reporters, or where activity was witnessed by others. This is not reasonable in case of isolated malicious behavior, such as denial-of-service attacks, and it is unclear how attackers would be prevented from labeling each other's reports. Our traffic proofs successfully eliminate false reports.

Publications [12], [14]–[16] propose various reputation models where voting is used to detect lying. This does not apply to our client reputations because of the isolated nature of malicious behavior and because attackers could participate in voting. It does however apply to detecting lying reputation centers.

VI. Future Work and Conclusions

While the design described here is convincing at the conceptual level, for it to provide an effective means for accountability in the future Internet, we need to solve many practical problems. Of critical importance is implementation of the proposed mechanisms in real hosts and routers to prove

that they can operate at packet speeds. This is our immediate next step.

Other real-world issues must also be considered. For example, our design, like many others, requires a ubiquitous PKI for bootstrapping source signatures. Such a PKI has never been accepted in today's Internet, and some of the reasons for that failure remain in the future Internet. For instance, there are administrative questions about who is allowed to provide signatures at the root of the PKI hierarchy. Such organizations have substantial control over Internet activities, so assigning this authority is an issue of great real-world importance. In our proposed system, the owner of the root of the PKI could make it impossible for entire segments of the Internet to provide accountability for their packets, likely with unpleasant consequences for users in that segment of the network. The problem is not unique to our proposed system, but must be resolved for the system to achieve its promise.

More thought is also required to deal with the issue of handling packets that come from sources that have been identified as malicious. We have already determined that long-term and short-term behavior make a difference in reputation, and that the weight servers put on a malicious behavior will depend on private criteria and on current server load. Other questions, including the type of the traffic or perhaps some kind of indemnification system, could bear on the issue of how to handle traffic from nodes that have behaved badly. A related issue is aggregation of reports of bad behavior, i.e. the exact algorithm that reputation centers use to compute a reputation score from bad reports. While reports are authenticated to prove that the client did indeed send the packets being complained of, we believe there is no lightweight, universally acceptable way to prove that the behavior was malicious. Sophisticated aggregation methods must be used to balance the necessity of a quick misbehavior punishment via lowering of a reputation score against a possibility of a bias.

Today's Internet is a difficult and mysterious environment, in large part because one can never be sure whether a particular packet, data stream, or application-level communication comes from its purported source. Full solution of this problem would allow Internet users to make others accountable for their actions. Proper, reliable attribution of network traffic to its originator is a precursor to any system of accountability. Attempts to retrofit such attribution onto today's Internet have failed. Building accountability into the core of any future version of the Internet is thus critical. This paper describes a promising approach to ensuring that, in the Internet of the future, every packet can be tied to the machine that sent it. This insurance allows both highly automated handling of ongoing attacks via tickets and reliable higher level analysis and long-term memory of host behaviors, via a reputation system. Jointly, source signatures, tickets and a client reputation system will provide the accountability today's Internet lacks.

REFERENCES

[1] Advanced Network Architecture Group. ANA Spoofer Project. http://spoofer.csail.mit.edu/.

[2] A. Yaar, A. Perrig, and D. Song. SIFF: A Stateless Internet Flow Filter to Mitigate DDoS Flooding Attacks. In *IEEE Symposium on Security and Privacy*, 2004.

[3] X. Yang, D. Wetherall, and T. Anderson. A DoS-limiting network architecture. In *ACM SIGCOMM*, 2005.

[4] V. Yegneswaran, P. Barford, and S. Jha. Global Intrusion Detection in the DOMINO Overlay System. Proc. the Network and Distributed Security Symposium (NDSS) 2004.

[5] D. Dagon, G. Gu, C. Zou, J. Grizzard, S. Dwivedi, W. Lee, and R. Lipton. A Taxonomy of Botnets. *Unpublished paper*, May 2005.

[6] P. Barford and V. Yegneswaran. An Inside Look at Botnets. *Special Workshop on Malwar Detection, Advances in Information Security, Springer Verlag*, 2006.

[7] D Andersen, H Balakrishnan, N Feamster, T Koponen, D Moon, and S Shenker. Accountable Internet Protocol (AIP) . In *Proc. of ACM SIGCOMM*, 2008.

[8] A. Shamir and Y.Tauman. Improved Online/Offline Signature Schemes. In *CRYPTO*, 2001.

[9] Xin Liu, Xiaowei Yang, David Wetherall, and Thomas Anderson. Efficient and Secure Source Authentication with Packet Passports. In *SRUTI*, 2006.

[10] Wolfgang Muhlbauer, Anja Feldmann, Olaf Maennel, Matthew Roughan, and Steve Uhlig. Building an AS-Topology Model. In *ACM SIGCOMM*, 2006.

[11] W. Xu and J. Rexford. MIRO: Multi-path Interdomain ROuting. In *ACM SIGCOMM*, 2006.

[12] K. Aberer and Z. Despotovic. Managing Trust in a Peer-2-Peer Information System. In Proceedings of the Tenth International Conference on Information and Knowledge Management, 2001.

[13] EBay. EBay Feedback Forum. http://pages.ebay.com/services/forum/feedback.html.

[14] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. Managing and Sharing Servers' Reputations in P2P Systems. In IEEE Transactions on Knowledge and Data Engineering, vol. 15, n.4, July/August 2003, pp. 840-854.

[15] M. Hou, X. Lu, X. Zhou, and C. Zhan. A trust model of p2p system based on confirmation theory. ACM SIGOPS Operating Systems Review, Volume 39 Issue 1, January 2005.

[16] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, P. Samarati, and F. Violante. A Reputation-based Approach for Choosing Reliable Resources in Peer-to-Peer Networks. In Proc. of the 9th ACM Conference on Computer and Communications Security, Washington, DC, USA, November 17-21, 2002.

[17] InformationWeek. Storm Worm Botnet More Powerful Than Top Supercomputers. http://www.informationweek.com/news/internet/showArticle.jhtml?articleID=201804528.

[18] B. Parno, D. Wendland, E. Shi, A. Perrig, B. Maggs, and Y. Hu. Portcullis: Protecting Connection Setup from Denial-of-Capability Attacks. In *Proceedings of the ACM SIGCOMM*, 2007.

[19] M. Natu and J. Mirkovic. Fine-Grained Capabilities for Flooding DDoS Defense Using Client Reputations. In *Proceedings of the Large-Scale Attack and Defense Workshop*, 2007.

[20] L. Wang, Q. Lu, and D. Luong. Engaging Edge Networks in Preventing and Mitigating Undesirable Network Traffic. In *Proc. of NPSec*, 2007.

[21] A. Bremler-Barr and H. Levy. Spoofing Prevention Method. In Proceedings of INFOCOM'05, March 2005.

[22] K. Park and H.Lee. On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law internets. In Proceedings of SIGCOMM 2001.

[23] J. Li, J. Mirkovic, M. Wang, P. Reiher, and L. Zhang. SAVE: Source Address Validity Enforcement Protocol. In *INFOCOM*, 2002.

[24] Z. Duan, X. Yuan, and J.Chandrashekar. Constructing Inter-Domain Packet Filters to Control IP Spoofing Based on BGP Updates'. Proceedings of INFOCOM'06, April 2006.

[25] C. Jin, H. Wang, and K.G. Shin. Hop-count filtering: an effective defense against spoofed DDoS traffic. Proceedings of the 10th ACM conference on Computer and communications security, 2003.

[26] A. Perrig, D. Song, and A. Yaar. StackPi: A New Defense Mechanism against IP Spoofing and DDoS Attacks. Carnegie Mellon University Technical Report, CMU-CS-02-208, February 2003.

[27] Mark Allman, Ethan Blanton, and Vern Paxson. An Architecture for Developing Behavioral History. In Proceedings of SRUTI 2005, pp 45-51.