A Poisoning-Resilient TCP Stack

Amit Mondal and Aleksandar Kuzmanovic
Northwestern University
{a-mondal, akuzma}@cs.northwestern.edu

***Abstract*— We treat the problem of large-scale TCP poisoning: an attacker, who is able to *monitor* TCP packet headers in the network, can deny service to all flows traversing the monitoring point simply by injecting a *single* spoofed data or control packet into each of the flows. One of the entities responsible for this severe vulnerability is certainly the TCP protocol itself: it behaves as a "dummy" state machine that can more-than-easily become desynchronized by an attacker. In this paper, we explore ways for upgrading TCP endpoints into viable DoS-resilient protocol entities, capable of mitigating large-scale poisoning attacks. We show, by means of analytical modeling, simulations, and Internet experiments, how small upgrades implemented by the endpoints can dramatically improve resilience to attacks. The key mechanisms unique to our approach are (i) *deferred protocol reaction*, used to accurately detect poisoning attacks; (ii) *forward nonces*, applied to distinguish among different traffic sources during the attack; and (iii) *self-clocking-based correlation*, utilized for successfully detecting legitimate packet streams. Our solution solely relies on the protocol design, it is incrementally deployable, and TCP friendly.**

I. Introduction

Denial of Service (DoS) attacks are presenting an increasing threat to the global inter-networking infrastructure. A troubling observation is that it is usually easier for the attacker to find a single security hole than for the defender to block all holes. However, reversing this asymmetric situation to the defenders' advantage is not impossible. Small efforts by defenders, typically easy to deploy and use, may raise the "bar" high enough, forcing the attacker to multiply the amount of resources in order to perform a successful attack.

The TCP's implicit assumption of end-system cooperation results in a well-known vulnerability to attack by high-rate non-responsive flows. However, to deny service to TCP traffic, it is not necessary to apply such high rate-attacks. Instead, an attacker who is able to monitor TCP packet headers in the network can send a *single* spoofed packet (*e.g.*, RST) to one of the TCP endpoints, and instantly poison the flow. As long as the spoofed packet's sequence number is in the acceptable range — the TCP endpoint will simply abort the connection [1].

This work addresses the problem of large-scale TCP poisoning attacks. An attacker who monitors traffic in the Internet can poison TCP flows traversing the monitoring point simply by injecting spoofed data or control packets to either the servers or the clients. In this way, the attacker can effectively deny service to both servers and clients, without directly dropping packets in the network, and without flooding network or server resources.

Many counter-DoS techniques, which address the problems of packet spoofing, header encryption, or authentication, directly or indirectly address the above problem, *e.g.*, [2]–[9]. Still, none of the solutions is widely deployed in today's Internet. As such, the approach we propose in this paper is a complementary effort aimed to increase the overall resiliency to DoS attacks. Moreover, our approach is unique in the sense that we apply no "classical" security techniques, but rather solely rely on DoS-resilient protocol design to defend against the attacks.

Our contributions are threefold. First, we demonstrate that by considering realistic limitations on the attacker, we open avenues for quite novel approaches to the poisoning problem. Second, we show that network measurements (latency in our case) can be used as an effective implicit authentication tool; the technique, we believe, might become a viable alternative for a number of other security-related problems. Finally, we demonstrate that the proposed scheme is effective in relieving the attacker from the ability to conduct simple, scalable, and low-rate attacks.

We start-off by recognizing the attackers' constraints: (i) Dropping or modifying packet is much harder to achieve than sniffing packets and (ii) identifying and utilizing lower latency Internet paths than those applied by the legitimate TCP endpoints requires a significantly larger amount of resources. Once these limitations are considered, the space of solutions significantly changes.

The first mechanism unique to our solution is deferred protocol reaction; instead of instantly processing the received packets, the TCP endpoints defer their reaction to incoming packets, which enables them to detect the attacks. For example, the arrival of data packets with the same or overlapping sequence numbers that carry different payload is a clear signature of the attack.

Next, we introduce forward nonces; the protocol requires the endpoints to repeat 8-bit long random numbers in successive TCP packets. This technique alone neither prevents attackers from generating meaningful spoofed packets, nor does it enable the receiver to authenticate arriving packets. Still, when combined with the attacker's limitation in utilizing lower-latency paths, it does provide the TCP endpoints with the

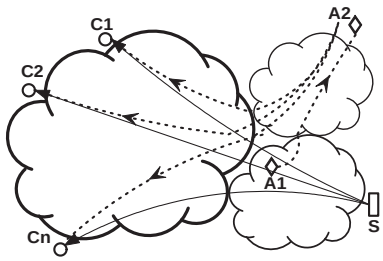

Fig. 1. TCP poisoning attack

valuable ability to distinguish streams generated by different sources.

Finally, we apply the self-clocking-based correlation mechanism to detect the legitimate flow. The key idea is to exploit TCP's self-clocking property, which induces high correlation between subsets of legitimate DATA and ACK streams. For example, if the attacker sends a single RST packet, the proposed mechanism can accurately detect the valid TCP flows by measuring high correlation between the appropriate substreams of DATA and ACK packets. Still, because dynamic network conditions can blur the correlation between the packet substreams, we perform an extensive simulation study to explore the robustness of the proposed techniques to diverse, often quite hostile, network conditions. To confirm the applicability of the self-clocking correlation approach in the Internet, we perform TCP measurements among several PlanetLab nodes [10] and find a high correlation between ACK and DATA packets measured along various Internet paths.

At the end, we explore the incremental-deployability properties of the poisoning-resilient TCP stack. The key problem is that due to delayed protocol reaction, the protocol utilizes less-than-TCP-friendly throughput. To resolve the problem, we derive a general TCP-friendly formula as a function of arbitrary additive-increase and multiplicative-decrease parameters α and β. Finally, we compensate the deferring effects by appropriately adjusting α and β. As a result, we demonstrate that in addition to achieving high resilience to poisoning, the proposed TCP stack remains TCP friendly in absence of attacks.

II. Motivation

A. A TCP-Poisoning Attack Scenario

What the attacker can do (part 1). Figure 1 depicts a TCP-targeted poisoning attack scenario. Victims, denoted by $C_1, \ldots, C_n$, are clients that download content from a server S. An attacker, denoted by A_1 (*e.g.* in the same subnet as the sender), monitors packets from the server to the clients. While this attacker could poison the TCP streams by sending bogus packets to either the clients or the server, this might not be feasible in reality; for example, because the attacker might not be able to send packets with forged source addresses. Further, attackers tend to avoid activity that is easily detected. Thus, we focus on a more discreet case of a cooperative version of the attack.[1] In a cooperative scenario, the attacker A_1 monitors only the victims' traffic and forwards valuable information (*e.g.*, source and destination addresses, together with port and sequence numbers of sampled packets) to another attacker, denoted by A_2 in Figure 1. A_2 then poisons the communication between the clients and the server by sending spoofed packets to either of the two.

Both TCP's control and data planes are equally vulnerable to the attack. For example, an attacker can subvert the TCP control plane by sending a reset (RST) packet to either the client or the server, thus immediately shutting down the connection. Likewise, the attacker can target the TCP data plane by sending bogus data packets to the receiver. A forged packet will be accepted by the receiving TCP agent and pushed to the application layer, as long as the forged packet is in the acceptable sequence-number range [1].

B. Attackers'Model

Here, we present the realistic constraints of the attacker. We will show below that these constraints can dramatically change the scope of possible solutions by opening the door to fundamentally novel approaches.

It is harder to modify or drop than to sniff packets. While the ability to modify or drop packets is justified in scenarios where the attacker compromises network routers (*e.g.*, [11]–[15]), or where hosts forward control and data packets on behalf of others, (*e.g.*, in multicast [16]–[18] or ad-hoc wireless networks [19], [20]), such capability is in general much harder to achieve in the Internet. On the other hand, monitoring packets is much easier; packet-sniffing software is freely available at various web sites and as commercial products [21], [22].

Limited "racing-success" probability. When the attacker sniffs a packet, assume it generates a spoofed packet, and send it to one of the endpoints. The probability that the spoofed packet will reach the destination before the valid packet, is significantly smaller than the probability that the valid packet reaches the destination first. Our arguments are the following. First, generating a spoofed packet itself may add a non-negligible delay, particularly when higher-layer packets must be generated. Second, if the valid and spoofed packets share the same network path to the destination, then unless the packets are re-ordered at routers, the packet generated first will reach the destination first. Finally, if the valid and the spoofed packets do not share the same network path (*e.g.*, Figure 1), it is indeed possible that the latency on the attacker's path (*e.g.*, A_1 — A_2 — C_1) can be shorter than the latency on the valid path (*e.g.*, A_1 — C_1). To achieve this the attacker needs to install its own overlay monitoring system. However, even

[1] Still, the solutions we develop later in the paper apply equally to the above scenario in which the monitoring machine (e.g., A_1) itself inserts packets to either the source or the destination.

in such case the probability of finding a lower latency path for the majority of clients (*e.g.*, $C_1,\ldots,C_n$) in the system is negligible [23].

What the attacker can do (part 2). In combination with packet spoofing, the monitoring capability can be utilized to conduct TCP-poisoning attacks, exactly as explained above. By sniffing a single packet, the attacker can send a single spoofed packet, and abort or poison the entire flow. Because the receiver windows are typically of the order of tens of packets [24], generating a spoofed packet with acceptable sequence numbers is highly feasible and requires no "hard" racing with valid packets. Moreover, once the attacker observes a DATA packet, it can poison the source by sending a spoofed ACK packet in the reverse direction; similarly, the attacker observing an ACK packet can poison the destination by sending a spoofed DATA packet.

III. Counter-DoS Techniques

A. Design Principles

Given that the attacker can only monitor packets, the stream of valid packets is continually present at the destination. We discuss exceptional scenarios (*e.g.*, Telnet-type applications) in Section VI. This view motivates a new way of approaching this problem: instead of trying to instantly authenticate the received packets, the goals are, first, distinguish among different streams, and then detect the valid one.

1) Deferred Protocol Reaction: Accurately detecting the TCP-targeted poisoning attack is the first step in addressing the problem. Given that the stream of valid packets will be present at the receiver, deferred protocol reaction to packet arrivals can be used to detect the attack. Delaying the protocol reaction for short intervals (exact values will be defined below) would be enough to receive contradicting messages, which would clearly reveal that the system is under attack. For example, the arrival of a RST packet followed by a stream of data packets would clearly reveal that the attack has been launched. Similarly, the arrival of data packets with the same or overlapping sequence numbers that carry a different payload would be a clear signature of the attack.

While quite effective in detecting poisoning attacks (as we will demonstrate below), deferred protocol reaction alone is insufficient to mitigate the attacks. The key problem remains: once the attack is detected, how to authenticate valid packets and drop the spoofed ones? We decouple the above problem in the following subproblems: (i) How to distinguish among the packets (or the streams of packets) generated by different sources? (ii) How to detect which of the sources is the legitimate one?

2) Forward Nonces: The first problem is distinguishing among packets generated by different sources. We propose forward nonces, which are similar in spirit, yet fundamentally different from the ones proposed by Savage *et al.* [25].

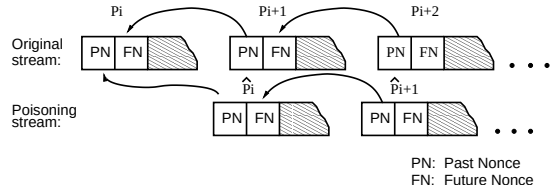

Fig. 2. Forward nonces (up) and concatenation attacks (down)

Forward nonces do not prevent the attacker from generating meaningful spoofed packets. Instead, they provide a simple chaining mechanism that enables distinguishing among different packet sources.

We introduce two new fields into the TCP packet format: *Past Nonce* and *Future Nonce*. Both are implemented as TCP options, as we elaborate below. Figure 2 illustrates this idea. For each DATA and ACK segment, the sender fills the *Future Nonce* field with a unique random number generated when the segment is sent. In addition, the sender fills the *Past Nonce* field with the random number that corresponds to the *Future Nonce* of the previous packet. Nonces generated by the source and the destination are independent of each other. Below, we explain how this method helps us separate packets generated by different sources.

What the attacker can do (part 3). Forward nonces impose a fundamental limitation on the attacker — it is no longer able to generate an easy attack by spoofing a packet with an arbitrary sequence number. Instead, to "break" the chain of valid packets, it must generate a packet with the nonce values relative to the sniffed packet. Moreover, nonces generated by the source and the destination are fully *independent* of each other. The implication of this design choice is significant: the attacker is no longer capable of injecting poisoning attacks in the direction opposite from the observed one, *e.g.*, send spoofed ACK packets after observing DATA packets, or vice versa. One exception are TCP SYN packets, which we discuss later in Section VI.

Assume the scenario from Figure 1. When the attacker A_1 sniffs a valid packet (*e.g.*, p_i) and accesses its payload and header values, including those of the *Past* and *Future Nonces*, it has the following options. First, the attacker can generate a spoofed packet (*e.g.*, $\hat{p}_i$) with exactly the same *Past* and *Future Nonces* as in p_i, and change either the payload or set the RST or FIN flags in the TCP header. However, as discussed in detail above (Section II-B), the spoofed packet $\hat{p}_i$ will reach the destination after the packet p_i with high probability. Hence, the attacked endpoint refuses the attack with the same probability simply by accepting the first of the two packets. Second, the attacker can launch the packet $\hat{p}_i$ with randomly generated *Past* and *Future Nonces*. However, because a randomly generated *Past Nonce* will not match the *Future Nonce* of the previous valid packet p_{i-1}, this type of attack is mitigated.

Finally, the best option for the attacker is to launch concatenation attacks, as illustrated in Figure 2. We explain the

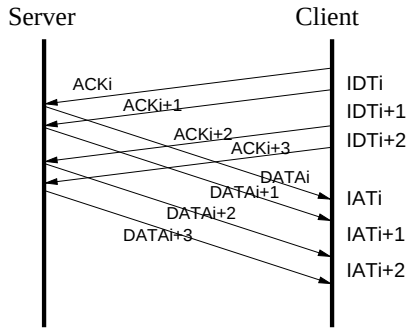

Fig. 3. Self-clocking-based correlation

single-packet attack version first. The attacker generates a spoofed packet (*e.g.*, $\hat{p}_i$) such that its *Past Nonce* matches the *Past Nonce* of the observed valid packet p_i; yet the *Future Nonce* is necessarily different; otherwise, we would repeat the scenario described above. Likewise, once the attacker sniffs the packet p_i, it can also generate a malicious packet $\hat{p}_{i+1}$, whose *Past Nonce* matches the p_i's *Future Nonce* (not shown in the figure). Still, the $\hat{p}_{i+1}$'s *Future Nonce* would again differ from the p_{i+1}'s with high probability, simply because these two are generated independently by two different sources. Finally, once the "chain" of valid packets has been "broken," nothing stops the attacker from generating streams of spoofed packets, as illustrated in Figure 2. We treat this problem in depth below.

In summary, forward nonces enable the destination to distinguish among legitimate and malicious (streams of) packets; yet, the important problem remains: which of the streams is the right one?

3) Self-clocking-based Correlation: Here, we propose a self-clocking-based correlation method as a way to accurately detect the legitimate TCP stream. This method is based on the TCP's self-clocking property, which induces high correlation between appropriate subsets of legitimate DATA and ACK packet streams.

TCP self-clocking characterizes the well-known TCP behavior in which the reception of ACK packets triggers the transmission of DATA packets at the sender [26]; likewise, the same term equally applies to the complementary scenario in which the reception of DATA packets at the receiver triggers the transmission of ACK packets. (We discuss the delayed ACK option of RFC 2581 [27] later in the text.) While the TCP self-clocking behavior is a consequence of the reliable window-based TCP congestion control, the key insight is that the timely responses to packet arrivals induce strong correlation between appropriate samples of inter-packet departure and arrival times at an endpoint, which we exploit to detect legitimate flows.

Figure 3 depicts a simple scenario showing the exchange of packets between a TCP sender and a receiver. We purposely idealize the scenario to convey the basics of the self-clocking correlation idea, and address many of the challenges later in the paper. Denote by IDT_i the inter-departure time between two consecutive ACK packets, ACK_i and ACK_{i+1}; likewise, denote by IAT_i the inter-arrival time between $DATA_i$ and $DATA_{i+1}$. As long as the sender transmits DATA packets in response to ACK packets (*e.g.*, $DATA_i$ in response to ACK_i, as shown in the Figure), and the packets are not significantly distorted in the network, the inter-ACK departure "code" set by the receiver will repeat in the inter-DATA arrival stream: IAT_i will be short (correlated to IDT_i), IAT_{i+1} will be longer (correlated to IDT_{i+1}), etc. Denote by N the number of inter-arrival and inter-departure samples. Then, we define the normalized distance between the two subsets, starting at index i and of the length N, $\sigma_i^N(IDT, IAT)$, as

$$\sigma_i^N(IDT, IAT) = \frac{1}{N} \sum_{k=i}^{i+N-1} \mid \frac{IAT_k - IDT_k}{IAT_k} \mid . \quad (1)$$

Computing the above metric over highly correlated packet substreams yields a small value, which can be used for implicitly authenticating the arrival streams. Likewise, higher distance values reveal potential attackers.

What the attacker can do (part 4). Nothing stops the attacker from launching longer-packet concatenation attacks. There are two options. First, the attacker may try to mimic the observed inter-packet times, thus achieving high correlation between legitimate and poisoned packet streams. However, endpoints can easily detect such attacks since poisoned packets will consistently reach the endpoints *after* the regular ones (Section II). Thus, the attacker is forced to apply a simple "see and shoot" strategy. Once the attacker observes a packet, it randomly intersperses concatenated packets around a reasonable, yet randomly chosen value. Hence, even if the attacker's packet arrivals are distributed around the correct mean value, they are unable to successfully mimic the inter-packet variations existent in the valid stream. As a result, the normalized distance for such streams increases, enabling the receiver to detect the attack. The implication is the following: even if the valid TCP transfer finishes while the concatenation attack is taking place, the TCP endpoint is able to accurately detect the malicious flow. This demonstrates the scheme's ability to thwart attacks the goal of which might be to prevent the connection to terminate by sending extra data packets.

B. Putting it all together

Here, we explain how we embed the above ideas into the TCP protocol. In addition, we define novel protocol parameters and TCP option fields, and provide guidelines for their settings.

1) How long to defer?: The deferring time parameter critically impacts an endpoint's ability to detect the poisoning attack. Setting it too low prevents successful detection; yet, setting it too high can unnecessarily degrade the protocol performance. Intuitively, relevant measures of interest here are the packet inter-arrival times; if the deferring time parameter is

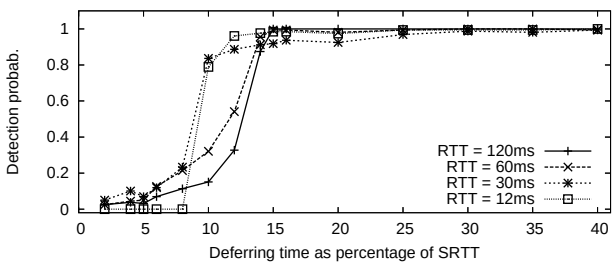

Fig. 4. Deferring time

set to a value that is longer than the packet inter-arrival times are, this guarantees detecting all attack attempts. Because the inter-arrival times are connection-dependent, the deferring time needs to be adjusted on per-connection basis.

To determine a reasonable value for the deferring time, we perform a number of simulations, and present the key results in Figure 4 (The exact simulation setup is explained in Section IV-A). The figure shows the attack detection probability as a function of the deferring time, which is expressed in terms of the percentage of the smoothed RTT (SRTT). As expected, the detection probability is poor for small deferring values. Yet, by increasing the deferring time parameter, the detection probability quickly increases. For example, Figure 4 shows that setting the deferring time parameter to 25% of the SRTT yields the detection probability above 0.99 for all evaluated RTT values. Thus, we opt for setting the deferring time parameter to 25% of the SRTT.

Deferred protocol reaction is tightly coupled with the self-clocking correlation mechanism. Despite delayed response to packet arrivals, an endpoint must preserve the self-clocking property characteristic of a non-deferring TCP. This could be achieved by equally delaying responses to packet arrivals. Hence, the deferring time parameter should not be updated over short time-scales because that can affect the accuracy of the self-clocking correlation method. This is exactly why we compute the deferring time as a function of the slowly-varying SRTT parameter. To avoid potentially frequent updates of the deferring parameter, we deploy additional low-pass filtering.

2) How to remain TCP-Friendly?: Delaying the protocol reaction by 25% of the SRTT by each of the deferring TCP endpoints effectively increases the TCP connection's RTT by 50%. This prevents such a connection to utilize the TCP-fair bandwidth share. According to [28], increasing the RTT by 50% approximately degrades the throughput by 33%. Still, this bandwidth loss could be easily compensated by retuning the additive increase and multiplicative decrease parameters α and β.

Using the stochastic TCP model and methodology of [28], we generalize the TCP-friendly formula to a scenario with arbitrary values of α and β. In particular, we express the average TCP rate *B* as a function of the round-trip time *RTT*, steady-state loss event rate *p*, TCP retransmission timeout value *RTO*, and number of packets acknowledged by each ACK *b*, as

$$\frac{1}{RTT\sqrt{\frac{2bp(d-1)}{\alpha(d+1)}} + RTO\min(1, 3\sqrt{\frac{bp(1+d)(d-1)}{2\alpha d^2}})p(1+32p^2)}. \quad (2)$$

We provide the derivation in [29].

Finally, by setting α=2 and β=0.54, we effectively undo the deferring effects. This helps the poisoning-resilient TCP to regain its TCP friendly fair-share, despite delayed protocol reactions. Later, in Section V, we demonstrate that this is indeed the case. To achieve the same for short TCP flows, we appropriately adjust the receiver window parameter and the TCP's slow-start behavior.

3) How Long are Nonces?: The nonce size determines the probability with which the attacker can launch a successful attack simply by guessing the *Future Nonce* field of the packet following the sniffed valid packet. If the guess is correct, and the malicious packet reaches the destination before the valid next-to-the-sniffed packet, the attack will be successful.

We propose using 8-bit long nonces, which we implement as a TCP option. Thus, the total overhead per packet is 2 Bytes, one for the *Past Nonce* and the other for the *Future Nonce*. With such an approach, the probability for the attacker to succeed by sending a single spoofed packet is 2^{-8}. At the same time, to guarantee the success of the attack, the attacker would have to send hundreds of Mbps bursts to the victim (*e.g.*, 2^8 packets over the interval of a few milliseconds in order to guarantee that atleast one of the attacker's packets will match the nonces).

4) How to Apply the Correlation Method?: Not all TCP packets are generated in response to incoming DATA or ACK packets. For example, whenever the TCP sender increases the window size, more than one DATA packet can be generated as a response to a single ACK. On the other hand, to effectively apply the self-clocking correlation ideas proposed above, a TCP endpoint must be capable of accurately filtering such packets. To cope with this problem, we propose that the endpoints set a single bit (implemented as a TCP option) only to packets generated directly in response to previously received packets, thus enabling a reliable use of the self-clocking method. While the use of a single bit is a type of explicit signaling between the two endpoints, it is here applied only to improve the performance of the implicit self-clocking correlation method, and not for explicit packet authentication.

Next, we explain the application of the self-clocking-based method, necessarily omitting many low-level details. Assume, for simplicity, a single-stream poisoning attack. Thus, whenever the attack is launched, the victim endpoint receives packets from two distinct sources, the malicious and the legitimate one. The self-clocking-based method is less reliable when the normalized distance is computed over short number of data points. Indeed, even a small deviation in the inter-

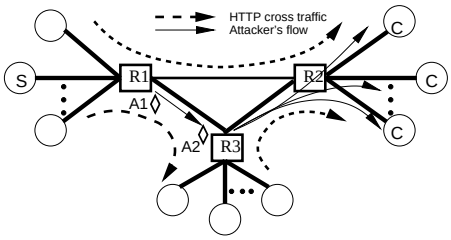

Fig. 5. Simulation scenario

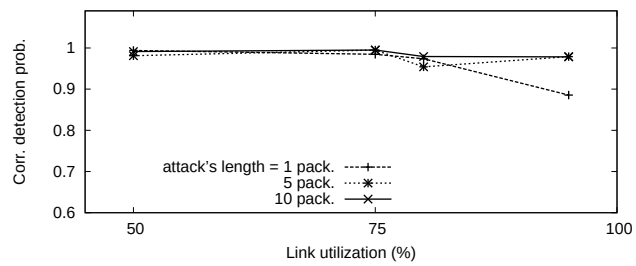

Fig. 6. Variable queuing delay

packet arrival time can cause errors. But, in the vast majority of scenarios, it is possible to bring reliable detection decisions based on a larger number of samples. For example, even if the attack is short, *e.g.*, 1-packet long, it is highly likely that a sufficient number of packets belonging to the legitimate stream will reach the destination after the start of the attack. We explore scenarios in which this is not the case later in the paper.

Thus, whenever one of the streams is less than 5-packets long, we compute the normalized distance defined by Equation (1), and compare it to a threshold. Numerous simulation experiments including scenarios with hundreds of flows, heterogeneous link capacities, and multiple bottlenecks, as well as Internet experiments corroborate that the threshold value of 0.8 represents a high performance compromise for all investigated scenarios. Finally, no threshold is needed when the number of packets from both streams is larger than 5, because we then directly compare the corresponding normalized distances and choose the one with smaller value.

IV. Measuring TCP-Poisoning Resilience

A. Simulation Scenario

Figure 5 depicts the simulation scenario. The topology consists of a web-client and a web-server pool that are interconnected by a pair of routers and a bottleneck link. Each node from the server pool connects to a router R1 with a 1 Gbps link; likewise, each node from the client pool connects to another router, R2, via a 1 Gbps link. Nodes R1 and R2 are connected by a link which capacity we vary from 10 Mbps (default) to 100 Mbps. By adjusting delays on the access links, we uniformly distribute the flow round-trip times in the range from 10 ms to 100 ms. We inject HTTP cross traffic on non-bottleneck links R1-R3 and R2-R3.

To simulate the distributed poisoning attack, initially illustrated in Figure 1, we proceed as follows. The attacker consists of two distinct entities — sniffing and poisoning. The sniffing entity, denoted by A1, monitors and forwards randomly-sampled packets to the poisoning entity A2 using IP-over-IP encapsulation. (Forwarding all packets is both not scalable and non-stealthy.) The attacker A2 obtains all relevant information by reading the TCP and IP headers of the sniffed packets. It then generates spoofed packets by forging the source address of the original packet within the acceptable sequence-number range of the receiver. The poisoning entity either launches a single-packet RST attack or a "see and shoot" concatenation attack. We consider five- and ten-packet long attacks. In both cases, we randomly intersperse attacker's packets in the range from 100 μsec to 3 ms.

To compute the attack-mitigation accuracy with low overhead, we initially do not abort a TCP connection once the attack is successful. In such cases, we simply increment the number of successful attacks, and continue with data transfer. Later, we do abort connections in order to evaluate the impact of the attack on throughput and fairness. Our implementation of the poisoning-resilient TCP is derived by modifying the *ns-2* FullTcpAgent stack. For every data sample, we run the simulation for 1000 sec repeatedly and take the average of the results.

B. Challenging Environments

Our anti-poisoning mechanisms exploit high correlation between subsets of inter-arrival and inter-departure times, which is induced by timely responses of legitimate TCP endpoints. However, such timely responses may become distorted due to queuing delay or packet loss, both of which are common in today's Internet. Below, we explore the behavior of our algorithm in such environments.

1) Variable Delay: Queuing delay and packet losses are correlated; for a given queue limit, the higher the queuing delay, the larger the packet loss probability. Our goal here is to understand the impact of both parameters on the detection accuracy. Thus, to decouple the two effects, we proceed as follows. First, to isolate the impact of the variable queuing delay, we increase the bottleneck queue limit for an order of magnitude, such that it becomes 25 times the bandwidth-delay product. In this way, bursts of highly variable HTTP cross traffic directly transfer into variable bottleneck queuing delay without causing packet losses. As a result, the inter-packet times of the TCP flows under attack become distorted, potentially weakening their resilience to attacks.

Figure 6 plots the correct detection probability as a function of the bottleneck link utilization, which we control by varying the cross traffic. For low link-utilization levels (*e.g.*, 50%), the queuing delay does not change dramatically, and thus the detection accuracy does not suffer. As the link utilization in-

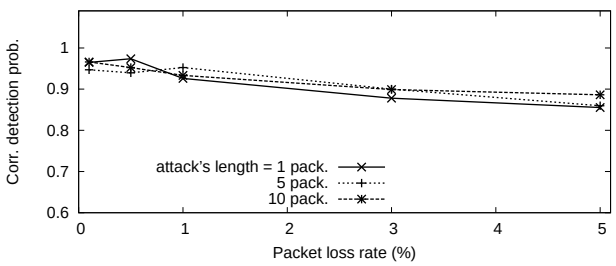

Fig. 7. The impact of packet loss ratio

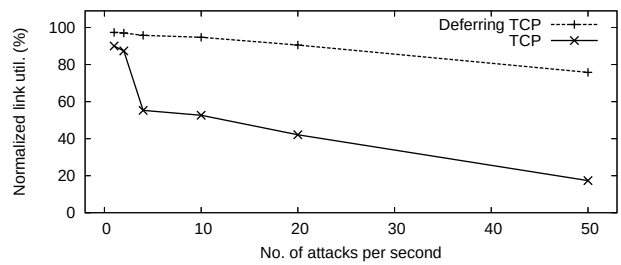

Fig. 8. Throughput

creases, so does the variability induced by the cross traffic, and the packets become more and more distorted. However, Figure 6 shows that the detection accuracy remains high, particularly for longer-packet attacks. Despite strong inter-packet aliasing, even short subsets of undistorted inter-arrival samples, existent in the legitimate TCP streams, are sufficient for distinguishing them from malicious streams, which lack such signatures. Finally, the correct detection probability degrades the most during single-packet attacks, when the distortion is largest (*e.g.*, 95% utilization). Indeed, in absence of longer malicious streams, we have to rely on the threshold-based scheme, which is less reliable in this case.

2) Congested Environments: To isolate the impact of packet losses on the accuracy of our detection scheme, we add an artificial packet dropper at the bottleneck link. In this way, we manage to control the packet loss rate, yet without increasing the queuing delay.

Figure 7 shows the correct attack-detection probability as a function of the packet loss rate, which we vary from 0.1% to 5%. As expected, the detection accuracy decreases as the packet loss ratio increases. Whenever a packet loss happens, the "chain" of valid packets breaks, which complicates the detection process. Once the attack is launched, the key parameter impacting its success is the number of valid packets arriving until the next packet loss event. If the number is high, there are almost no differences from the loss-less case, and the detection probability is high. However, as the number of valid packets that can be used for detection decreases, the detection accuracy worsens, as discussed in the previous section. Finally, if the attacker generates a spoofed packet that matches a packet loss event, the attack is certainly successful. Still, the probability for this to happen is small, necessarily smaller than the packet loss probability.

In summary, highly congested network environments do degrade the anti-poisoning detection accuracy. Luckily, such environments do not prevail in today's Internet. Recent network measurements reveal that a large percentage of TCP connections (*e.g.*, 20%) experience no packet loss, while only a negligible percent of connections (*e.g.*, 0.06%) experience a loss rate of more than 10% [30]. Below, we re-evaluate this result by performing Internet experiments.

3) The Impact on Throughput: Thus far, the goal was to estimate the correct detection probability in various environments, so we did not abort the connections in the case of a successful attack. Here, we evaluate the effects of poisoning on throughput; thus, we reset TCP connections whenever the attack is successful. To create a realistic networking environment, we generate the traffic by randomly distributing file sizes in the range from 10 kB to 10 MB. [2] To support a larger number of flows, we increase the bottleneck capacity to 100 Mbps.

Figure 8 plots the normalized link throughput as a function of the RST attack rate, which varies from 1 to 50 RST packets per second. In the first scenario, we evaluate the performance of the regular TCP stack (marked as "TCP" in the figure). For even moderate attack rates, *e.g.*, 4 RST packets per second, the link utilization drops almost by a half. Indeed, as long as the spoofed packet's sequence number is in the acceptable range (easy to achieve), the TCP endpoint aborts the connection [1]. As the attack rate increases, the normalized throughput further quickly decreases. It does not drop all the way to zero due to high arrival and departure rates of short flows.

At the same time, the "deferring TCP" stack remains highly resilient to attacks, despite a large number of short flows. For example, in the most severe scenario (RST rate equals 50 packets per second), and despite large number of short flows, the throughput remains high, approximately four times higher than in the regular TCP scenario. One interesting effect we observed is that whenever the attack starts succeeding and resetting a percent of flows, the congestion reduces; as a result, the self-clocking technique becomes more accurate, such that the throughput remains high.

C. Internet Experiments

Here, we perform Internet experiments to evaluate the accuracy of the self-clocking-based detection method in a real system. We do *not* generate any attacks; instead, we simply measure the normalized distance between DATA and ACK packets, which indirectly reveal the potential attack-mitigation probability.

[2]Our results, not shown due to space constraints, indicate high resilience of both short and longer-lived flows to poisoning attacks. Also, a poisoning attack against short flows quickly experiences scalability limitations.

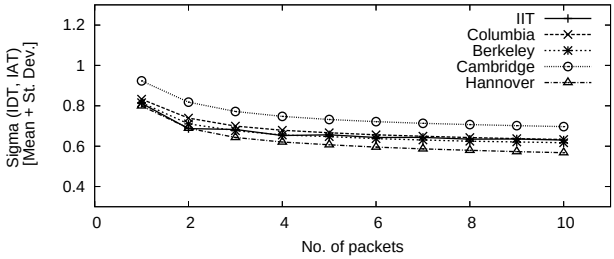

Fig. 9. Internet experiments

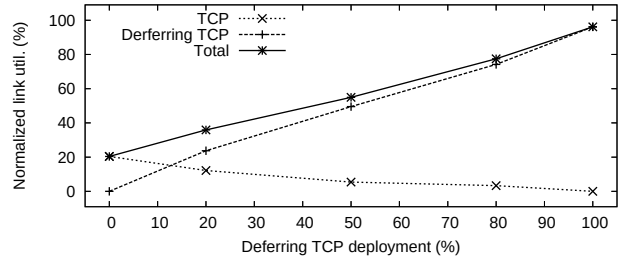

Fig. 10. Incremental deployability; in presence of attacks

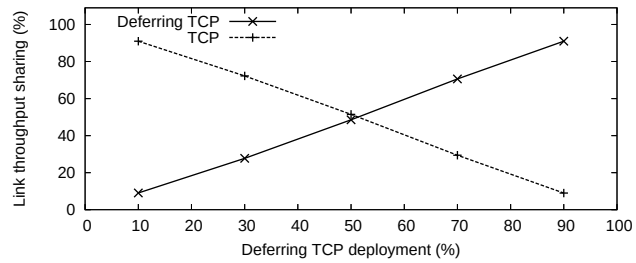

Fig. 11. TCP fairness in absence of attacks

We establish TCP connections from a host machine located at Northwestern Campus to 5 PlanetLab nodes around the world [10]: IIT (India), Columbia (New York), Berkeley (California), Cambridge (England), and Hannover (Germany). We perform number of measurements by transferring 5 MB files from the host to the destinations; the transfers last from 40 sec to Berkeley up to 400 sec to IIT. In all scenarios, we measure the normalized distance, $\sigma_i^N(IDT, IAT)$, defined by Equation (1); in this case, the IDT sequence corresponds to DATA packets generated by the host machine, while the IAT corresponds to ACK packets returned by destinations. We compute normalized distance over *all* substreams of the size i, ranging from 1 to 10.

Figure 9 depicts the mean plus one standard deviation of $\sigma_i^N(IDT, IAT)$ as a function of i for all measured Internet paths. As expected, the normalized distance decreases as the length over which the measure is taken increases. This is primarily because the variance of the normalized distance decreases. However, the key insight from the figure is that while slightly larger than in the simulations, the normalized distance between DATA and ACK substreams is pretty much the same. Moreover, in all scenarios, mean plus one standard deviation are below the threshold of 0.8 for 5 packet-long intervals. This result confirms that the proposed method is indeed highly accurate in detecting low-rate poisoning attacks.

One of the reasons for the slight increase in the normalized distance between DATA and ACK packets is the delayed ACK feature. A TCP endpoint may delay transmission of ACKs hoping to have DATA ready to sent in that frame. In any case, for obvious reasons, poisoning-resilient TCP simply does *not* apply the delayed ACK feature, but requires endpoints to reply immediately.

V. Incremental Deployability

In this section, we treat the problem of incrementally deploying deferring TCP in the Internet. In essence, we explore how regular and deferring TCP streams affect each other when they are multiplexed.

A. Performance in Presence of Attacks

Figure 10 plots the normalized link utilization as a function of the percentage of deferring TCP connections in the system. The plot clearly reveals that as the number of poisoning-resilient TCP flows in the system increases, the total link utilization increases. In presence of attacks, regular TCP flows' service is easily denied, whereas deferring TCP survives the attack; hence, the link utilization increases. Because the TCP connections are in this scenario limited by the receiver window, the link utilization does not increase quicker. In other circumstances, deferring TCP flows would be able to utilize bandwidth left unused by regular TCP flows, thus providing an additional incentive for clients to apply the novel TCP stack.

B. Performance in Absence of Attacks: **TCP Fairness**

Here, we explore the fairness properties of the proposed TCP stack. Typically, such experiments are needed to show that the new stack does not overwhelm the regular TCP. However, in our case, the goal is opposite. Due to deferred protocol reaction, the proposed TCP effectively increases the RTT by a half. To compensate for these effects, it also adjusts the protocol parameters as explained in Section III-B.2. Figure 11 shows that the compensation is indeed successful, since the protocol remains TCP friendly. We conduct an experiment with a large number of long-lived TCP flows which create congestion. Independently of the level of deployment, deferring TCP flows manage to utilize their bandwidth fair share. Finally, this also confirms the correctness of our non-trivial modeling efforts, which resulted in Equation (2).

VI. Discussion and Related Work

A. Discussion

Forward vs. "classical" nonces. Savage *et al.* [25] propose *Nonce* and *Nonce reply* fields as a way to prevent a misbe-

having receiver from driving a standard TCP sender arbitrarily fast. For each segment, the sender fills the *Nonce* field with a unique random number, which is echoed by the receiver in the *Nonce Reply* field. This is fundamentally different from the *forward nonces* mechanism proposed here. The key difference is that in our scenario, nonces generated by the two endpoints are independent of each other. Also, the TCP endpoint that generates a nonce is *itself* required to repeat the same nonce in the successive TCP packet. This enables the destination endpoint to distinguish packets generated by different sources. In addition, the "classical" nonces ([25]) address a different problem, and does not solve the TCP-poisoning problem: the attacker can return a correct *Nonce Reply* field, yet maliciously set the RST flag. Moreover, nothing prevents the attacker from sending a future data packet within the receiver window, yet with a different nonce.

Interactive communication. All proposed counter-poisoning mechanisms critically depend on the assumption that the sender is backlogged, *i.e.*, always has packets to send. Because interactive applications (*e.g.*, Telnet) violate this assumption, the proposed solutions do not directly apply to such scenarios. While beyond the scope of this paper, one way to address the problem would be sending "dummy" packets *at low rates* into the network in moments when no data is coming from the application.

Poisoning TCP SYN packets. Using independent nonces in different flow directions (*i.e.*, source-to-destination and vice-versa) prevents the attackers from generating meaningful poisoning attacks towards the origin endpoints (*e.g.*, returning ACK packets to the source after observing DATA packets). However, this does not hold for the exchange of initial (*i.e.*, TCP SYN) packets, in which case the attacker can start a meaningful concatenation attack towards the source. While this effectively becomes a hijacking attack, our approach still has a great potential to combat the problem. First, due to deferring, we are *always* capable of detecting such attacks. Moreover, applying additional techniques (*e.g.*, estimating the actual RTT to the other endpint either by sending active out-of-band pings or by using a history-based approach [31]) could be used to effectively defend against such attacks.

B. Related Work

Our approach relies solely on DoS-resilient protocol design, and requires no "classical" security techniques. Here, we briefly summarize such techniques. The following system vulnerabilities enable the TCP poisoning attack: (i) the lack of an authentication mechanism between senders and receivers, (ii) the visibility of TCP packet headers in the network, and (iii) the attacker's ability to generate packets with forged source addresses.

An authentication mechanism would prevent the TCP poisoning attacks. One such approach is proposed in [3] to defend against TCP-based BGP-targeted attacks. It defines a new TCP option for carrying an MD5 digest in a TCP segment [32]. This digest acts like a signature for that segment, incorporating information known only to the connection endpoints. There are several drawbacks to such an approach. First, the computation burden of such algorithms may become a system bottleneck on high-bandwidth networks [8]. Second, the key exchange and management (*e.g.*, required by [33]) is itself an unsolved problem [15]. Building an Internet-wide public key infrastructure (PKI) incurs huge costs and suffers from a high risk of failure [34], [35]. Finally, independently of the PKI problem, the initial key-exchange (*e.g.*, based on the *Diffie-Hellman key agreement* [36]) is itself vulnerable to poisoning attacks: an attacker that observes the transfer of a public key is unable to decrypt messages encrypted by that key, but nothing stops the attacker from poisoning the initial transfer of keys, *e.g.*, by resetting a TCP (or a lower-layer) connection before the keys are exchanged.

The IPsec protocol [4] encrypts the TCP packet headers and payload. Thus, users applying the IPsec are immune to the TCP-targeted poisoning attacks. However, there are several drawbacks with such an approach. First, while IPsec improves users' security and privacy, it also increases general network vulnerability to DoS attacks. For example, without the ability to monitor packet headers and classify packets, counter-DoS and intrusion-detection systems (*e.g.*, [37]) simply cannot function. Second, the inaccessibility of packet header bits in the network prevents deployment of advanced transport protocols (*e.g.*, XCP [38]) as well as novel security mechanisms for BGP (*e.g.*, [15]), which require explicit flow monitoring. Third, IPsec is incompatible with widely-deployed network address translators (NATs) [39].

Finally, preventing malicious hosts from sending spoofed packets would also solve the TCP-targeted poisoning problem. One approach is ingress filtering in which ISPs on the edges drop outgoing packets with forged source addresses to mitigate DoS attacks. However, ingress filtering has not been widely deployed for economic reasons: ISPs must pay for a system that only benefits others. Moreover, even if ingress filtering were universally deployed at the customer-to-ISP level, attackers could still forge addresses from the hundreds or thousands of hosts within a valid customer network. Another approach is IP traceback [5]–[7], [9]. Such mechanisms, when implemented at network core routers, can detect hosts that forge source IP addresses. Unfortunately, such mechanisms either require routers to keep a large amount of state [7] or generate a large amount of overhead traffic [5]. In addition, problems such as scalability, incremental deployability, and large hardware changes required at routers further prevent the deployment of traceback mechanisms in the Internet [9].

VII. Conclusions

This paper addresses the problem of large-scale TCP-poisoning attacks, in which an attacker can severely deny service to a large number of flows by poisoning the endpoints with spoofed packets. We design and evaluate a poisoning-

resilient TCP stack, which applies novel mechanisms, (i) deferred protocol reaction, (ii) forward nonces, and (iii) self-clocking-based correlation to accurately detect, distinguish, and mitigate poisoning attacks. We demonstrate that the proposed TCP upgrades relieve the attacker from the ability to conduct simple, scalable, and low-rate attacks, in which even a single spoofed packet is sufficient to deny service to a flow. To succeed, the attacker is forced to flood the endpoints, thus becoming detectable by other counter-DoS mechanisms.

The proposed solution requires no explicit security association between the TCP endpoints, nor it requires them to explicitly prove the receipt of packets. Instead, the legitimate TCP endpoints challenge and authenticate each other implicitly by recognizing random "codes" embedded in the inter-packet departure and arrival sequences. Large-scale simulation and Internet experiments show remarkably high accuracy of this scheme in diverse, even quite hostile, networking environments. The proposed TCP upgrades are incrementally deployable; clients applying the change become resilient to attacks while they experience no performance degradations in their absence, since the protocol is TCP friendly. In the future work, we plan to implement the proposed TCP stack and validate its performance in a controlled network testbed as well as on the Internet.

VIII. Acknowledgements

This work is supported by NSF CT grant ANI-0627715.

References

[1] "Transmission control protocol," Sept. 1981, Internet RFC 793.

[2] P. Ferguson and D. Senie, "Network ingress filtering: Defeating denial-of-service attacks which employ IP source address spoofing," May 2000, Internet RFC 2827.

[3] A. Heffernan, "Protection of BGP sessions via the TCP MD5 signature option," Aug. 1998, Internet RFC 2385.

[4] S. Kent and R. Atkinson, "Security architecture for the Internet protocol," Nov. 1998, Internet RFC 2401.

[5] J. Li, M. Sung, J. Xu, and L. Li, "Large-scale IP traceback in high-speed Internet: Practical techniques and theoretical foundation," in *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, May 2004.

[6] S. Savage, D. Wetherall, and T. Anderson, "Network support for IP traceback," *IEEE/ACM Transactions on Networking*, vol. 9, no. 3, pp. 226–237, June 2001.

[7] A. Snoeren, C. Partridge, L. Sanchez, C. Jones, F. Tchakountio, B. Schwartz, S. Kent, and W. Strayer, "Single-packet IP traceback," *IEEE/ACM Transactions on Networking*, vol. 10, no. 6, pp. 721–734, Dec. 2002.

[8] J. Touch, "Performance analysis of MD5," in *Proceedings of ACM SIGCOMM '95*, Boston, MA, Aug. 1995.

[9] A. Yaar, A. Perrig, and D. Song, "FIT: Fast Internet traceback," in *Proceedings of IEEE INFOCOM '05*, Miami, FL, Mar. 2005.

[10] "Planetlab," http://www.planet-lab.org.

[11] S. Cheung, "An efficient message authentication scheme for link state routing," in *Proceedings of the Annual Computer Security Applicataions Conference*, San Diego, CA, Dec. 1997.

[12] Y. Hu, A. Perrig, and D. Johnson, "Efficient security mechanisms for routing protocols," in *Proceedings of NDSS '03*, San Diego, CA, Feb. 2003.

[13] Y. Hu, A. Perrig, and M. Sirbu, "SPV: Secure path vector routing for securing BGP," in *Proceedings of ACM SIGCOMM '04*, Portland, Oregon, Sept. 2004.

[14] O. Nordstrom and C. Dovrolis, "Beware of BGP attacks," *ACM Computer Comm. Review*, vol. 34, no. 2, pp. 1–8, Apr. 2004.

[15] L. Subramanian, V. Roth, I. Stoica, S. Shenker, and R. Katz, "Listen and whisper: Security mechanisms for BGP," in *Proceedings of NSDI '04*, San Francisco, CA, Mar. 2004.

[16] C. Karlof, N. Sastry, Y. Li, A. Perrig, and J. Tygar, "Distillation codes and applications to DoS resistant multicast authentication," in *Proceedings of NDSS '04*, San Diego, CA, Feb. 2004.

[17] A. Perrig, R. Canetti, B. Briscoe, J. Tygar, and D. Song, "Efficient authentication and signing of multicast streams over lossy channels," in *Proceedings of the IEEE Symposium on Security and Privacy*, Berkeley, CA, May 2000.

[18] A. Perrig, R. Canetti, D. Song, and J. Tygar, "Efficient and secure source authentication for multicast," in *Proceedings of NDSS '01*, San Diego, CA, Feb. 2001.

[19] Y. Hu, D. Johnson, and A. Perrig, "SEAD: Secure efficient distance vector routing for mobile wireless ad hoc networks," *Ad Hoc Networks*, vol. 1, no. 1, pp. 175–192, July 2003.

[20] Y. Hu, A. Perrig, and D. Johnson, "Packet leashes: A defense against wormhole attacks in wireless networks," in *Proceedings of IEEE INFOCOM '03*, San Francisco, CA, Apr. 2003.

[21] ProgramURL.com, "Packet sniffing software," http://www.programurl.com/software/packet-sniffing.htm.

[22] OptOut, "Packet sniffing," http://grc.com/oo/packetsniff.htm.

[23] D. Anderson, H. Balakrishnan, M. Kaashoek, and R. Morris, "Resilient overlay networks," in *Proceedings of ACM SOSP '01*, Banff, Canada, Oct. 2001.

[24] A. Medina, J. Padhye, and S. Floyd, "Measuring the evoluation of transport protocols in the Internet," Tech. Rep., 2004.

[25] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson, "TCP congestion control with a misbehaving receiver," *ACM Computer Comm. Review*, vol. 29, no. 5, pp. 71–78, Oct. 1999.

[26] V. Jacobson, "Congestion avoidance and control," *ACM Computer Comm. Review*, vol. 18, no. 4, pp. 314–329, August 1988.

[27] M. Allman, V. Paxson, and W. Stevens, "TCP congestion control," Apr. 1999, Internet RFC 2581.

[28] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Reno performance: A simple model and its empirical validation," *IEEE/ACM Transactions on Networking*, vol. 8, no. 2, pp. 133–145, Apr. 2000.

[29] A. Kuzmanovic and E. Knightly, "Receiver-centric congestion control with a misbehaving receiver: Vulnerabilities and end-point solutions," *Journal of Computer Networks*, 2007.

[30] M. Allman, W. Eddy, and S. Ostermann, "Estimating loss rates with TCP," *ACM Performance Evaluation Review*, vol. 31, no. 3, Dec. 2003.

[31] T. Anderson, A. Collins, A. Krishnamurthy, and J. Zahorjan, "PCP: Efficient endpoint congestion control," in *Proceedings of NSDI '06*, San Jose, CA, May 2006.

[32] R. Rivest, "The MD5 message-digest algorithm," Apr. 1992, Internet RFC 1321.

[33] R. Moskowitz, P. Nikander, P. Jokela, and T.Henderson, "Host Identity Protocol," June 2006, Internet draft draft-ieft-hip-base-06.txt.

[34] D. Davis, "Compliance defects in public key cryptography," in *Proceedings of the USENIX Technical Conference*, San Diego, CA, Jan. 1996.

[35] C. Ellison and B. Schneier, "Ten risks of PKI: What you're not being told about public key infrastructure," *Computer Security Journal*, vol. 16, no. 1, pp. 1–7, Apr. 2000.

[36] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. IT, no. 11, pp. 644–654, Nov. 1976.

[37] R. Mahajan, S. Floyd, and D. Wetherall, "Controlling high-bandwidth flows at the congested router," in *Proceedings of IEEE ICNP '01*, Riverside, CA, Nov. 2001.

[38] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," in *Proceedings of ACM SIGCOMM '02*, Pittsburgh, PA, Aug. 2002.

[39] B. Adoba, "IPsec-NAT compatibility requirements," May 2001, IETF Internet draft.