Counteracting Discrimination against Network Traffic

Ioannis Avramopoulos, Jennifer Rexford
Princeton University

Dimitris Syrivelis, Spyros Lalis
University of Thessaly

ABSTRACT

End users, edge networks, content providers, and service providers alike all need effective ways to counteract *traffic discrimination*—the selective (mis)treatment of packets as they flow through the Internet. However, preventing discrimination, or even detecting ongoing discrimination, is difficult in practice. Instead, our solution (which we call *surelinks*) forces the discriminator to introduce more easily detectable loss and delay, and then moves traffic away from offending paths. Surelinks combine three techniques: encryption of aggregate traffic between edge nodes, multipath routing to circumvent performance problems, and stealth probing for accurate measurements in an adversarial setting. Experiments with our prototype system, implemented in the Click router, demonstrate that surelinks have practical overhead comparable to destination-based forwarding.

1. INTRODUCTION

Traffic discrimination involves dropping, delaying, or modifying data packets based on their contents. The discrimination may be performed by a router or firewall along the path, or by using routing-protocol attacks to direct the packets to an end host for further processing. Traffic discrimination comes in many forms, as the following examples illustrate:

Censorship: Censorship involves filtering or modifying traffic based on packet headers (e.g., IP addresses or TCP port numbers) or the payload. Common examples include Web-filtering software and the censoring of politically-sensitive material by the "Great Firewall of China." As another example, the Canadian ISP Telus blocked access to Web sites supporting a strike of the Canadian Telecommunications Workers Union against Telus [1].

Differentiated services: ISPs may offer differentiated services by placing high-priority traffic in the "gold queue," based on the packet headers, as a paid service for their customers. Alternatively, an ISP may charge content providers a fee to avoid having their traffic receive poor performance in the "bronze queue." These fears about traffic discrimination lie at the heart of the ongoing "net neutrality" debate, where content providers have advocated (unsuccessfully, thus far) legislation to prevent discrimination [2].

Biasing measurement results: Monitoring services like Keynote collect measurements to rate ISPs in terms of performance. ISPs have an incentive to give preferential treatment to the measurement traffic to improve their rankings, leading Keynote to respond by employing "anti-gaming" techniques [3]. ISPs also have an incentive to give preferential treatment to the active probes their customers send to verify service-level agreements.

Malicious attacks: Anecdotal evidence suggests that router compromises, either from remote attackers or disgruntled network operators, have become an operational reality [29]. Once in control of a router, the adversary may configure packet filters that drop or delay a subset of the traffic. Alternatively, the adversary may launch a routing-protocol attack to attract the traffic to an end host under his control. Either way, the adversary may *selectively* drop traffic to target a particular victim or to evade detection.

In this paper, we propose techniques for counteracting discrimination, regardless of the motives of the discriminating party.[1] Encryption clearly has an important role to play in preventing discrimination. For example, the Peacefire anticensorship software [4], released as early as 1996, uses encryption to thwart Web-filtering firewalls. Encryption makes traffic classification more difficult, forcing the discriminator to either spend more resources to classify traffic or apply coarse-grained discrimination rules that make his actions easier to detect. However, encryption does not entirely prevent discrimination. Classification may still be possible through traffic analysis (e.g., analyzing packet sizes and timing), as in Narus' techniques for identifying Skype traffic [31]. Furthermore, as a proactive technique, encryption alone cannot react to ongoing discrimination; instead, encryption should be complemented by reactive techniques that circumvent discriminating devices.

One seemingly natural solution is to automatically detect discriminating paths or devices and route around them. However, we argue that detecting discrimination is both *hard* and, in fact, *unnecessary.* Detecting discrimination is fundamentally difficult without a reference model of "neutral" behavior. Because the end users and applications accessing the Internet receive variable levels of service in an unpredictable fashion even during normal operation, a precise definition of "neutrality" has remained elusive despite significant efforts to formalize it [17]. Instead, we advocate a solution that does *not* require detecting discrimination. In particular, we force the discriminator to create easily-measurable packet loss or delay, and move traffic to alternate paths when measurements reveal bad performance.

Our solution, called *surelinks (secure virtual links)*, has two main ingredients:

- **Encryption of aggregated traffic:** Packets are directed via an encrypted tunnel between edge routers (or end hosts). This forces the discriminator to resort to coarse-grained discrimination that triggers measurable performance disruptions on the end-to-end path.
- **Multipath routing:** Upon detecting bad performance, the edge node can select an alternate path for the surelink (if multiple paths exist) or direct traffic on an overlay path consisting of two or more surelinks. Accurate

[1]Note, however, that we use the terms *discriminator* and *adversary* interchangeably.

measurements, even when an adversary is trying to bias the results, are important. Our *stealth probing* protocol achieves this goal by sampling traffic using a keyed hash of the packets at each end of a surelink.

In Section 2, we present the threat model. Section 3 describes how surelinks use encryption of aggregated traffic and multipath routing to counteract discrimination. Then, Section 4 describes how stealth probing provides secure availability monitoring. Section 5 presents our prototype implementation of surelinks in the Click platform. The evaluation of our prototype in Section 6 illustrates that surelinks can run at high speed, even on a software router. We discuss related work in Section 7 and conclude the paper in Section 8.

2. THREAT MODEL

In this paper, we present techniques that counteract an adversary, possibly located in-between the communicating parties, who tries to selectively degrade the performance of, or entirely prevent, communication between end systems. More precisely, we say that a path is *available* if end hosts can make use of this path to communicate. According to this definition, the availability of a path can be determined by the performance requirements of the end hosts and the performance characteristics of the path. Common performance characteristics are the loss rate, delay, jitter of the delay, and available bandwidth. In affecting availability, the adversary may affect any of these quantities so that the availability requirements of the end hosts are not met—our focus in this paper will be on the loss rate and delay.

The simplest form of discrimination involves a network device (e.g., a router) blocking a subset of the packets while allowing the remaining packets through. More broadly, discrimination aims to differentiate service quality by classifying traffic based on source and destination addresses, port numbers, or payload content and treating the classes preferentially. Discrimination requires the ability to inspect the traffic either through wiretapping or through the control of network equipment such as a router or middlebox. Traffic may also be intercepted through routing-protocol attacks. Discrimination also requires the ability to affect the service offered to the traffic. The service can be affected by discarding, modifying, or degrading the performance of the traffic. Discarding traffic that crosses a router is straightforward by installing packet filters in the data plane. Modifying traffic may involve the formation of *data wormholes* through the deflection of victim traffic to remote hosts (such as network scrubbers or botnets). Degrading performance may involve the configuration of low-priority queues, the formation of data wormholes, or remote denial-of-service attacks targeting links of the communication path.

3. SURELINKS: SECURE VIRTUAL LINKS

A surelink counteracts discrimination through a combination of proactive and reactive countermeasures. The proactive countermeasure is an encrypted tunnel between hosts or routers that carries a mix of traffic. The encrypted tunnel makes traffic classification hard and forces the adversary to inflict more collateral damage on non-targets. The reactive countermeasures reroute traffic away from badly-performing paths, by picking an alternate path through the network or directing traffic through a sequence of surelinks. The individual mechanisms have been applied in different contexts, including encrypted tunnels in virtual private networks [39] as well as techniques for rerouting in non-adversarial settings (e.g., intelligent route control [19] and resilient overlay routing [10]). In our work, we combine these two mechanisms, along with secure availability monitoring, to allow surelinks to counteract discrimination in an adversarial setting.

3.1 Proactive: Encrypting Aggregated Traffic

Encryption is a first line of defense against discrimination. The simplest form of traffic classification, and the one employed by the majority of firewalls, is by inspection of the string of bits that comprises a packet; encryption makes this string of bits, in part, unintelligible to the firewall. The information that encryption hides depends on the layer in the network stack where it is applied:

Encrypting the application data: Encryption can be performed between the transport and application layers, as in TLS [15]. Although the application payload is protected, the discriminator can still drop or delay packets based on the transport-layer protocol and port numbers, or the IP addresses of the communicating end-points.

Encrypting the transport header: Encryption can be performed between the network and transport layers, as in IPsec [24]. Hiding the port numbers helps conceal the application the end-points are running. However, the IP addresses of the communicating end-points are still visible, and the discriminator may be able to identify the application based on packet sizes and timing.

The information that encryption hides also depends on the network device (host or middlebox/router) performing the encryption. Encrypting *aggregated* traffic makes traffic analysis harder, and forces the adversary to resort to coarse-grained discrimination that is easier to detect. Routers or middleboxes can direct traffic through an encrypted tunnel that hides the IP addresses of the communicating end-hosts and mixes traffic from a variety of users. Directing traffic through a network *relay* can also hide the addresses of the communicating endpoints. For example, some anti-censorship systems allow hosts to bypass a firewall by communicating indirectly through a *circumventor* [4, 5] using TLS to encrypt the traffic. Using IPsec is more appealing than TLS in this scenario as well, since TCP-based tunnels are ill-suited for certain applications such as streaming.

In our design of surelinks, we use IPsec—we decided against TLS for the aforementioned reasons. Nodes (hosts or edge routers) direct packets over an IPsec tunnel that carries a mix of traffic. The edge nodes establish the tunnel in an automated fashion, using DNS and the Internet Key Exchange (IKE) [20]. DNS enables the automated discovery of hosts and surelink-compliant network devices. IKE is used to establish security associations with authorized devices. Tunnel establishment is discussed in greater detail in [13].

3.2 Reactive: Selecting Alternate Paths

Multipath routing, coupled with accurate performance measurements, can counteract discrimination by circumventing the discriminating network devices. Multipath routing is based on a virtual topology of surelinks created by a group of hosts and routers. Surelinks connect *relay points* located in the hosts and routers. Traffic is directed over paths with higher availability either by following one of multiple underlying paths between a pair of relay points or by following virtual paths consisting of a sequence of surelinks.

Moving traffic to alternate (non-discriminating) paths is feasible only if the network has sufficient path diversity. Fortunately, many users and edge networks connect to the Internet in multiple locations, often through different upstream providers. Although each connection offers a different path to the destination, these paths sometimes have nodes and links in common; support for multipath interdomain routing [37] would offer even greater path diversity. Alternatively, the end hosts or edge routers could create additional diversity by forming an overlay network that directs traffic through intermediate nodes [4]. Furthermore, a wireless broadband user can shift traffic to a different upstream provider by cooperating with other wireless users [9]. For example, if user A experiences poor performance accessing destination X, A could use the wireless network to tunnel traffic through user B's machine. Each of these techniques offers end users and edge networks significant opportunities to circumvent discriminating paths.

Using multipath routing to circumvent discrimination relies on accurate monitoring of path performance. These measurements must be robust to a discriminator that tries to evade detection by preferentially treating the measurement traffic while simultaneously dropping or delaying the regular data packets. Although end users are the definitive judges of communication quality, manual detection of discrimination, as is the case today in, say, the Peacefire system, can be a tedious process. Automating detection can lead to more effective and scalable countermeasures against discrimination. In the next section, we describe how surelinks can apply stealth probing, our automated technique for measuring path performance in the presence of adversaries.

4. SECURE AVAILABILITY MONITORING

Robust measurements of path performance are crucial for driving the routing decisions for surelinks. In contrast to traditional intelligent route control (IRC) schemes that assume a benign network, we must ensure that an adversary cannot bias the measurement results to make round-trip paths he controls look better than *better* paths he doesn't control. For example, if path p, free from the adversary, experiences 5% packet loss and path q, controlled by the adversary, experiences 25% packet loss, the adversary should not be able to trick the measurement process to prefer q over p. After discussing the limitations of existing measurement techniques, we describe how stealth probing can prevent the adversary from biasing measurement results. Then, we describe how to tune the sampling rate and discuss why stealth probing operates at the network layer.

4.1 The Case for Passive Probing

Existing measurement techniques are either *active* (sending probe packets and observing their performance) or *passive* (collecting statistics for existing traffic). Both approaches have serious limitations in an adversarial setting, arguing for an alternative approach we call "passive probing."

Active measurement involves sending probes from one node to another using ICMP packets (e.g., echo requests and replies), UDP packets (e.g., DNS queries and responses), or TCP packets (e.g., HTTP downloads). Active measurement is heavily used in IRC systems that reroute traffic to circumvent failures and network congestion, especially since active probes can measure alternate paths that are not currently in use by regular data traffic. However, active probes introduce extra packets that consume resources along the path—the probes may, in fact, interfere with the very properties they are designed to measure. In addition, the probes must be designed carefully to ensure they observe the same statistical properties that the real traffic does, which is quite challenging in practice [14].

Perhaps more importantly, an adversary on the data path can easily distinguish probe packets from normal traffic, and treat the probes preferentially (e.g., by delivering the probes while discarding or delaying non-probes). The probe packets may be identified based on their protocol number (e.g., ICMP) or IP addresses (e.g., if they correspond to dedicated probing servers), as well as other information such as their size or timing. Concealing the probes by, for example, keeping the identities of probing servers secret, would imply the compromise of security once these identities are leaked either through traffic analysis or other means. In fact, any attempts to achieve security by the secrecy of the identities of the probing machines would be a direct violation of the Kerckhoffs' principle that "the enemy knows the system." We do not expect that active probes mounted from hosts can be made indistinguishable while still retaining their ability to measure the path performance of the data traffic (which precludes employing overlay-based anonymizing techniques) without the direct involvement of the routers. Furthermore, securing active probes by having routers add padding and randomize the timing of probes, while helpful, would increase measurement overhead and decrease accuracy.

Passive measurement observes the existing traffic traversing a link. As a result, passive measurement is typically used to collect statistics about the traffic load and application mix, rather than to observe path performance. One exception is Listen [35], which passively monitors the progress of TCP flows to identify reachability problems. However, solutions like Listen require per-flow state and cannot distinguish between server and network failures; in addition, in the absence of encryption, an adversary could conceivably impersonate the destination to evade detection. Another passive monitoring scheme is Trajectory Sampling [16], which observes the same packets at multiple monitoring locations. However, sampling must be done carefully to ensure that the adversary does not know which packets serve as implicit probes. In addition, using trajectory sampling to drive routing decisions requires a way to provide feedback to the ingress node about path performance.

Instead, we advocate a hybrid solution we call "passive probing" in which we select (or *sample*) a subset of the existing packets to serve as *implicit* probes. To provide feedback to the ingress node, the remote end-point of the surelink sends an acknowledgment packet in response to each sampled packet. In contrast to active probing, passive probing does not introduce much extra traffic because of the small size of the acknowledgments. Furthermore, passive probing enables performance inference robust to adversaries through the concealment of probes and a performance inference methodology that is based on statistical sampling theory. To provide performance measurements for all paths, the surelink end-points split traffic over multiple paths; path splitting is attractive for a variety of other reasons, including more flexible load-balancing policies and limiting the ability of a single adversary to see all of the traffic.

4.2 Stealth Probing

Stealth probing is a *secure* passive probing protocol that relies on the following three processes:

A **sampling process** selects a subset of the packets crossing the path to serve as implicit probes. The output of this process becomes available to both the ingress end-point, which expects an ACK for each probe, and the egress end-point, which replies with an ACK. By keeping a timestamp for each probe, the ingress end-point can measure both packet loss and round-trip delays.

A **concealment process** prevents an on-path adversary from distinguishing between probe and non-probe packets. This process precludes the preferential treatment of probe packets by the adversary.

An **integrity assurance process** prevents an adversary from inducing inaccurate measurements by modifying traffic, injecting forged traffic, or replaying old traffic (either data packets or ACKs).

To ensure that both end-points select the same packets as probes, the sampling is *pseudo*random, using a hash function. The ingress and egress end-points apply the same hash function to a subset of bytes in the data packet, ignoring the bytes that change as the traffic traverses the network (e.g., the IP checksum and TTL fields) or have very low entropy (e.g., the IP version number), as in trajectory sampling [16]. If the image of the hash falls below a predetermined threshold, the packet is treated as an implicit probe; otherwise, the packet is a not a probe. The concealment process is implemented by choosing the hash function in a way that prevents the adversary from learning the hash. For example, the end-points could sample using a keyed hash function or encrypt all of the data packets to ensure the adversary cannot distinguish between probe and non-probe traffic. Integrity assurance is achieved through authentication of data packets and acknowledgments.

For each passively-sampled data packet, the egress end-point sends an acknowledgment back to the ingress end-point. In theory, we could reduce the overhead of stealth probing by having the egress end-point send periodic reports about a *window* of probe packets instead of per-probe acknowledgments. However, periodic snapshots would lead to slower detection of packet losses and would prevent accurate estimates of round-trip times, making it difficult for the ingress end-point to react to attacks (such as data wormholes) that introduce delay. Although explicit acknowledgments introduce overhead, these packets are much smaller than the average data packet, limiting the overhead.

Stealth probing measures for each sample the sum of the delay of the corresponding data packet from ingress to egress plus the delay of the acknowledgment in the reverse direction. It is important to note that the egress end-point does *not* conceal the acknowledgment through any sort of encryption, padding, or randomized timing. Therefore, the adversary can give preferential treatment to acknowledgments so that they experience different delay than the data packets in the direction from egress to ingress. However, by dropping or delaying an acknowledgment, the adversary would only make the round-trip path performance look worse, leading the surelink end-points to select an alternate path. Furthermore, the margin the adversary has for improving the delay of the acknowledgments is upper bounded by a small value equal to the queuing delay in the adversary's routers of the path. Also note that since stealth probing provides round-trip performance measurements, the two end-points of the surelink may need to coordinate when they choose alternate paths. This is especially important when the paths are asymmetric, since an adversary on the reverse path may have an incentive to discredit the performance of a benign forward path by selectively dropping ACK packets.

4.3 Performance Inference from Samples

Because stealth probing samples packets essentially randomly, we can use statistical sampling theory [36] to compute tight statistical bounds on the estimation error vis-a-vis exhaustive measurements. In this section, we use this theory to answer the question of how low to set the sampling rate in stealth probing so as to achieve a target statistical accuracy. Rather than present a definitive sampling rate, we present a methodology for making informed decisions about it according to the requirements of individual networks.

We will illustrate the methodology with a specific example. Consider a pair of networks jointly deploying stealth probing in the pair of their adjoining paths (forward/reverse). The network operators must make a decision about which value to set the sampling rate to. This decision involves a trade off between the benefit and cost of monitoring.

Suppose that the objective of the network operators in deploying stealth probing is to determine whether the loss rate of the round-trip path exceeds a loss-rate-threshold λ_0, raising an alarm if it is above. (A packet is considered lost if an ACK is not received or if the round-trip delay exceeds a delay-threshold.) I.e., the monitor must decide between two hypotheses, H_0, the loss rate is below λ_0, vs. H_1, the loss rate is above λ_0 using the results of the probes. There are two types of possible errors. If H_0 is falsely rejected, there is a *false alarm.* If H_1 is falsely rejected, there is a *miss.* Finding an algorithm to decide between H_0 and H_1 involves a trade-off between the probabilities of the two types of errors, since one can always be made arbitrarily small at the expense of the other. The Neyman-Pearson criterion [27] for balancing the errors is to place an upper bound on the false-alarm probability and, given this bound, to minimize the miss probability. The choice of a false-alarm probability involves a trade off between the benefit and cost of monitoring. As the false-alarm probability increases, the benefit of monitoring decreases since the value of the information contained in an alarm decreases. However, reducing the false-alarm probability requires expending additional resources in taking measurements, increasing the cost of monitoring. The choice of a false-alarm probability is situation-specific.

Given the false-alarm probability, the network operators can decide on a sampling rate by an iterative process. According to this process, the sampling rate is successively increased and the corresponding impact on precision is gauged. Given a sequence of trials, network operators can decide on a value for the sampling rate based on the desired precision. More specifically, given a false-alarm probability and a sampling rate, an algorithm for the hypothesis-testing problem can be devised. In the example, it is possible to use an optimal (*uniformly most powerful* [27]) algorithm. In Fig. 1, we plot the probability of false alarm as a function of the number of losses in a window of packets for increasing values of the sampling rate. We assume that the probability of a false alarm is 5%, the observation window is $10,000$ packets, and the loss-rate threshold is 10% (corresponding to 1000 packets). We observe that there are diminishing returns by increasing the sampling rate and that there is little

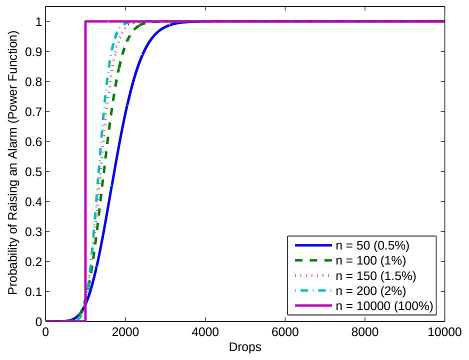


Figure 1: Probability of an alarm as a function of the number of losses for different sampling rates.

value in increasing the sampling rate above 2%.

4.4 Network-layer Monitoring

Our design of a secure availability monitor places the key monitoring functions at the *network* layer of either a host or a network device (middlebox/router). An interesting alternative would be to have applications perform their own monitoring, to enable greater customization of the measurements according to application-specific requirements. Such an approach would resemble current practices of users manually detecting the loss of availability but would take such practices one step further by trying to automate the process. However, relying on application developers to incorporate customized monitors would add a dependency that could delay adoption of discrimination countermeasures by the end users. Furthermore, the majority of IP traffic is generated by a narrow set of applications, such as Web, e-mail, VoIP, and P2P. Therefore, we argue that monitoring does not need to be highly customizable to be effective.

Placing the network-layer monitoring at end hosts is attractive as it frees the users from dependencies on third parties, such as application developers and network providers, who might in fact oppose the deployment of discrimination countermeasures. However, placing monitors at edge routers is an even more attractive option for the following reasons: First, the edge router sees a much higher volume of traffic, enabling fast, accurate detection of performance problems. In contrast, each host sees only a small subset of the traffic, making detection slower. Second, the edge router can react to the measurement results by shifting traffic away from a faulty path. Furthermore, in shifting traffic to new paths, the edge router can manage the aggregate traffic demands with the organization's load-balancing policies in mind. In contrast, end hosts typically do not have sufficient visibility or control to take corrective action. Transferring control from the organization's network to the end hosts would lead to extra overhead and additional security risks from compromised hosts. Third, although detection of a path failure by an edge router monitoring a path to another router naturally implies that this path should be avoided, detection of a path failure by a host monitoring a path between hosts is ambiguous as to whether a router in the path or the traffic's destination has failed. Finally, monitoring between edge routers requires only one security association for each pair of stub networks. In contrast, managing security associations for all pairs of end hosts would be complicated.

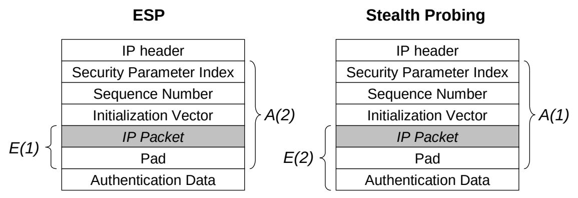


Figure 2: Packet format of ESP in tunnel-mode. E stands for encryption and A for authentication. ESP performs encryption before authentication (left). Stealth probing performs the reverse (right).

5. IMPLEMENTATION

We prototyped surelinks to evaluate their performance and make them available for use in operational systems. We used the the Click [25] platform for building software routers. Open-source routers serve as research platforms [8] and are also offered as commercial products, typically deployed at enterprise networks. We used the Click framework for building software routers because, by comparison to the native Linux support for routing, Click has better support for extensibility. This enabled us to efficiently implement data forwarding functionality critical to the performance of our system (such as the integration of IPsec security policy with data forwarding outlined in Section 5.3) in a straightforward fashion. Our implementation of IPsec tunnels has been incorporated in the standard Click distribution.

A Click router is a set of interconnected packet processing modules running in a kernel thread. These modules, also called *elements*, are C++ objects. Elements and their interconnections can be represented by a directed graph, also called a Click *configuration*, that characterizes the processing flow of packets. Surelinks were implemented using modules that extend an existing IP router Click configuration. In the rest of this section, we present how we implemented surelinks and also our implementation choices.[2]

5.1 Concealment and Integrity Assurance

The encryption and authentication required for concealment and integrity assurance are implemented using the Encapsulating Security Payload (ESP) protocol [23] of IPsec in tunnel mode, which provides end-to-end cryptographic protection at the IP layer. In tunnel-mode ESP, IP packets are encapsulated by an ESP header and trailer containing cipher-specific fields. The ESP packet is, first, tunneled by encapsulating it in a new IP header (see Fig. 2), followed by the application of symmetric ciphers to the encapsulated packet. ESP uses a 32-bit sequence number as part of a mechanism that protects from replay attacks. This mechanism is also used in surelinks to protect from attacks that introduce delays so as to reorder packets.

Our implementation of ESP is fully compliant with the ESP standard with the exception that authentication is performed before encryption (see Section 5.2). We used the AES encryption cipher and the HMAC-SHA1 authentica-

[2]For the interconnection of Click modules in surelinks and a summary of the control flow, the reader may refer to [13].

tion cipher; code for those ciphers was obtained from the OpenSSL toolkit.

5.2 Sampling Based on the Keyed Hash

Several possibilities exist for implementing the sampling process. One possibility is to hash immutable parts of the inner IP packet (before encryption at ingress and after decryption at egress) using, for example, modular division as in trajectory sampling [16]. In order to avoid the additional overhead from modular operations, we chose a different implementation of the sampling process that leverages the pseudorandom output of the keyed hash that authenticates the packet. Using the authenticator, sampling can be performed by comparing its value (two bytes in our implementation) to a threshold. However, ESP sends the authenticator in the clear (see Fig. 2), making it easy for the adversary to distinguish the probes. We address this by reversing the order of encryption and authentication (see Fig. 2). If encryption is performed in Cipher Block Chaining (CBC) mode, performing authentication before encryption has been shown to be secure [26]. Because SHA1 behaves like a random function, the sampling probability is the same for each packet, resulting in an unbiased selection of probes.

5.3 Integrating Security Policy & Forwarding

Security policy specifes what traffic should be protected by the surelinks and how. This information must be accessible in an efficient and scalable manner. IPsec defines a Security Policy Database (SPD) to store this information at the ingress router through which policy is enforced using packet filters. Rather than implement a separate SPD, we integrate its functionality into the Forwarding Information Base (FIB) of the router, possibly obviating the need for additional filters. The FIB stores the outbound interface per destination prefix and must be looked up for inbound packets to make a forwarding decision. We extend the FIB to support SPD functionality by adding an additional field to each FIB entry. This field stores a pointer to the IPsec security association (SA) for the corresponding destination prefix or NULL, if surelinks have not been configured for that prefix. In this way, irrespective of the number of outbound IPsec tunnels, the SA for encapsulating a packet is retrieved simply by following the pointer of the FIB entry. Configuring surelinks at a granularity finer than the existing prefixes in the FIB can be achieved by installing FIB entries for more specific prefixes (as the *longest prefix match rule* selects the most specific entries). Access control lists that match the five-tuple can provide even finer control.

Encapsulation at the ingress router and decapsulation at the egress router should be performed using the same SA. However, the egress router must decapsulate the packet before accessing the FIB, making the technique used by the ingress router inapplicable. In order to retrieve the SA, the egress router maintains a Security Association Database (SADB) accessed using the Security Parameter Index (SPI), a value that is stored in the ESP header of the incoming packet. The SPI is a 32-bit integer that, together with the address of the egress router, uniquely identifies the SA.

The only parameter that can affect the SA lookup delay, and, thus, the scalability of the design, is the granularity of security policy. Adding more prefixes in the FIB when subprefixes require separate SAs will increase the FIB lookup delay. Installing access lists will increase the packet processing delay. Noticeable delays of those kinds could only arise in a widespread deployment, in which case operational practice would likely suggest a preferable method for implementing granular policies. The number of inbound tunnels at an egress router only affects the size of the hash table that stores the SAs.

5.4 Consistent Traffic Splitting

Splitting inbound traffic to multiple outbound paths is preferable to directing all traffic to a single path for several reasons; it provides the ability to monitor traffic using passive probing, to implement flexible load-balancing policies, and to limit the traffic any single adversary would see. Traffic splitting is usually performed using a hash function such as CRC16: The range of the hash function is divided into intervals, where each interval corresponds to an outbound path, and hashing maps inbound packets to the intervals. Most routers hash the source and destination IP address fields of the packet; hashing the five-tuple is also supported by some routers. In the general case, traffic is split into unequal proportions (decided by a load balancing algorithm such as [18]). We consider here the simpler case of an even traffic distribution implying equal-length intervals. This case may arise in practice if stealth probing makes a binary decision to either use or avoid each path.

Even in this simple case, the aforementioned traffic-splitting technique is known to perform poorly when paths are added or deleted dynamically [21]. The reason is that additions and deletions typically result in significant unnecessary changes to the mapping between paths and TCP flows that may cause the disruption of the flows. Consistent hashing [22] can minimize changes when objects (such as web pages) are mapped to a dynamic set of bins (such as web caches). We have implemented a traffic splitting scheme for mapping inbound IP traffic to outbound paths based on consistent hashing that we evaluate in the next section. To our knowledge, this is the first use of consistent hashing as an IP data forwarding module.

5.5 Performance Monitoring

Upon sampling a packet, the egress router responds by sending an ACK to the ingress router. To enable the ingress router to match the ACK with the corresponding probe, the ACK includes the SPI of the tunnel and the 32-bit sequence number of the probe. We also authenticate the ACK using the SA of the probe, which prevents (on-path or off-path) adversaries from forging it. Receiving and generating ACKs is handled by the corresponding Click kernel threads at the ingress and egress routers, respectively, to avoid the overhead of using a separate process.

Timing by the Kernel: Measuring losses and round-trip delays requires timing the probes and ACKs at the ingress router. Keeping a timeout per probe would meet the timing requirements of the measurements and at the same time make the result immediately available for monitoring performance. However, this method would incur significant processing overhead for scheduling and cancelling timeouts and because of the frequent interruption of the kernel thread. We implemented a different timing method that can decrease the CPU load by adding configurable delays to the availability of measurement results. According to this method, the data forwarding modules are only responsible for logging and timestamping probes and ACKs while losses and round-trip delays are computed by reading the logs. Logs are read by a Path-Performance Monitor (PPM) that has been implemented in user space. We elaborate on the PPM later in this section.

The interface between the kernel and PPM is based on a *circular buffer* that essentially acts as a queue between them. The circular buffer is exported as a regular file using a Linux proc-like filesystem called *clickfs*. There are two circular buffers, one for the probes and one for the ACKs. An applicable size for each buffer can be determined by the sampling rate, the packet rate, and the round-trip time. For example, if the bit rate of incoming traffic is 1Gbps and the average packet size is 500B, then the packet rate is 250000pps. Assuming also a sampling rate of 2% and a round-trip time of 1sec, the log should contain about 5000 entries. Each log entry contains the 32-bit SPI, the 32-bit sequence number, and a 64-bit timestamp. Therefore, 5000 entries correspond to 80KB of memory.

Monitoring in User Space: The Path-Performance Monitor (PPM) is a daemon implemented in Java that reads the probe- and ACK-logs in order to measure losses and round-trip delays. PPM should be invoked frequently enough to prevent *rollover* of the circular buffers that would imply loss of measurement data. Rollover could be prevented by forced context switches from the kernel to PPM that we decided against because they are intrusive. The kernel adjusts instead the priority of the PPM daemon according to the occupancy levels of the buffers and warns PPM after a rollover.

PPM processes the logs using the following algorithm: First, the entries of the probe-log are added to a hash table. Then, for each entry in the ACK-log the hash table is queried for a matching probe. If a probe is found, the round-trip delay is calculated and the probe is removed from the hash table. Otherwise, the ACK is considered spurious and it is silently discarded. If all entries in the ACK-log have been processed and there are remaining probes in the hash table, the probe timestamps are compared to the current time for detecting possible losses.

6. EVALUATION

In this section, we present an evaluation of our prototype implementation in terms of the latency that surelinks add on top of IPsec/ESP and destination-based forwarding, the increase in CPU load, and the resulting throughput penalty for input traffic of different size. The experiments show that surelinks perform comparably to IPsec and destination-based forwarding for a realistic traffic mix.

6.1 Experimental Setup

Our testbed consists of four commodity PCs connected in sequence by three Ethernet links ($H_1 \rightleftharpoons R_1 \rightleftharpoons R_2 \rightleftharpoons H_2$). Nodes R_1 and R_2 run Click 1.5.0 and serve as routers, while H_1 and H_2 serve as hosts. We set the MTU on links $H_1 \rightharpoonup R_1$ and $R_2 \leftharpoonup H_2$ to 1400-bytes in order to avoid fragmentation due to IP and ESP encapsulation. Propagation delay is negligible and the links are used exclusively by these nodes, i.e., there is no cross traffic.

Each link is implemented using a crossover cable connected to a dedicated Ethernet interface. Each PC has a 2.8 GHz Pentium-4 processor with 1 GB of RAM and running a Linux 2.6.16 kernel. We perform our experiments both for 100Mbit (Realtek) and 1Gbit (D-Link) Ethernet cards. The Ethernet device drivers used in our setup do not support "device polling", a technique used by Click for improving performance. The effects are noticeable in the 1Gbit interfaces. Device polling would generally improve throughput performance but it would not significantly affect our comparisons. Note also that despite having full-duplex links, routers cannot send and receive packets simultaneously as they are limited by the DMA (Direct Memory Access) chip, which can process at most one transfer request at a time. This negatively affects the throughput performance of stealth probing which produces additional traffic in the opposite direction (i.e., ACKs).

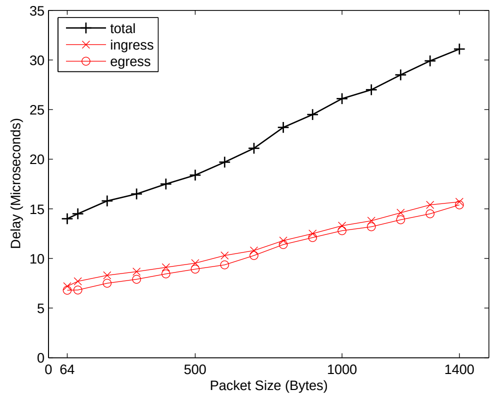


Figure 3: Processing delay stealth probing adds to a non-probe packet as a function of the packet size.

6.2 Processing Delay

Figure 3 shows the processing delay added by surelinks at the ingress and egress routers to a non-probe packet as a function of the packet size. This overhead is exclusively due to IPsec/ESP processing, because stealth probing only introduces an additional comparison operation.

Delays were measured using the Click profiler to record the cycle counts of 50 echo requests sent from H_1 to H_2 going through the tunnel from R_1 to R_2. We report the the minimum[3] values at ingress and egress and their sum. Stealth probing adds a total processing delay on top of IP that varies between 14 and 31 μsec, depending on the packet size. The increase in delay for larger packet sizes is due to additional processing required by the authentication and encryption ciphers.

If a packet is selected to serve as a probe, there is an additional delay of 2.5 μsec at ingress (to update the log) and 8.2 μsec at egress (to generate the ACK), giving a total of 10.7 μsec, irrespective of packet size. Notably, this delay occurs only x% of the time, where x is the sampling rate.

6.3 Data Throughput

To assess the effect of surelinks on data throughput, we measure the Bulk Transfer Capacity (BTC) which corresponds to the achievable throughput of a bulk transfer TCP connection [34]. For this purpose, we have configured an Iperf client on node H_1 and an Iperf server on node H_2 and

[3]Our measurements has small variations of the order of 0.1 μsec due to operating system activity.

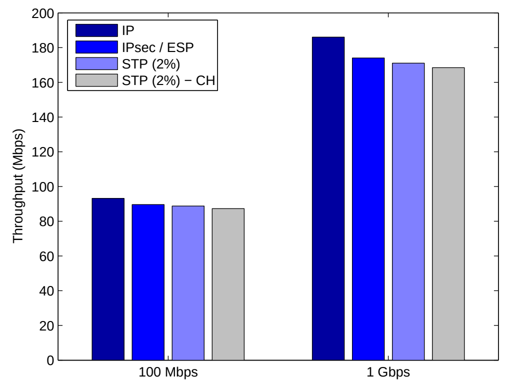


Figure 4: Data throughput with 100Mbps and 1Gbps links.

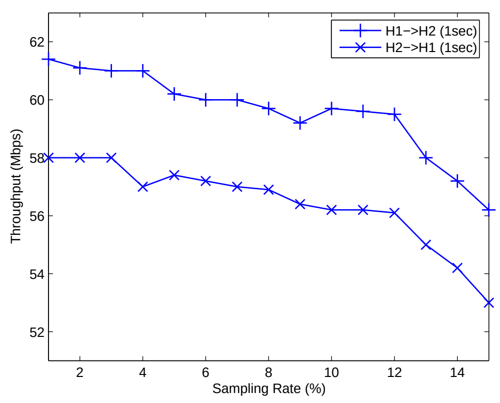


Figure 5: Impact of sampling rate on TCP throughput.

let the client send data to the server over a TCP at full speed. Each experiment had a duration of 20 sec and was repeated ten times; here we report the average values.

Fig. 4 shows the data throughput for four different configurations between R_1 and R_2: IP, IPsec/ESP, stealth probing at 2% sampling rate, and stealth probing (at 2%) with consistent hashing (selecting one among 100 tunnels). We show measurements for 100 Mbps and 1 Gbps links. In the first case, the stealth probing data rate is 5% less than IP and 1% less than IPsec/ESP, whereas in the second case, the stealth probing data rate is 8% less than IP and 2% less than IPsec/ESP. In both cases, consistent hashing further decreases the data rate by 2%. As a note, stealth probing at zero sampling rate (no packets are selected as probes and no ACKs are generated) had the same performance as IPsec/ESP.

To measure the impact of the sampling rate (and ACK packets) on bi-directional data throughput, we use an additional Iperf client and server pair to send data in the opposite direction. Both TCP sessions begin simultaneously and have a duration of 20 sec. Like before, each experiment is repeated ten times, and we report the average of the obtained measurements. The link bandwidth for these experiments is set to 100 Mbps.

Fig. 5 shows the data throughput of the two TCP sessions as the sampling rate varies from 1% to 15%. We observe that up to a 10% sampling rate, throughput decreases slowly as the sampling rate increases; despite the linear increase in ACK-traffic and the additional processing overhead for generating and reading the logs. We attribute this to the small size of stealth-probing ACKs (80-bytes) and the efficiency of our performance monitoring scheme.

6.4 CPU Resources

Figure 6 shows CPU utilization during the experiments of the previous section that involve a single Iperf client-server pair and TCP session. These figures were obtained using the *top* utility. In comparison to IP, IPsec increases the CPU utilization by 13% for 100 Mbps links and 26% 1 Gbps links. In comparison to IPsec, stealth probing increases the CPU utilization by 2% in both cases. Consistent hashing further increases the CPU load by 2% in both cases. We observe that most of the CPU resources are consumed in packet forwarding. In routers with high-speed links, packet forwarding would typically be assigned to dedicated hardware.

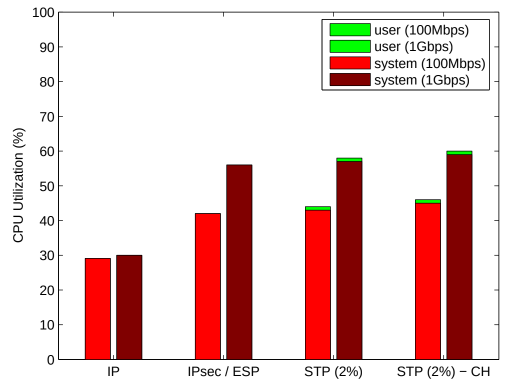
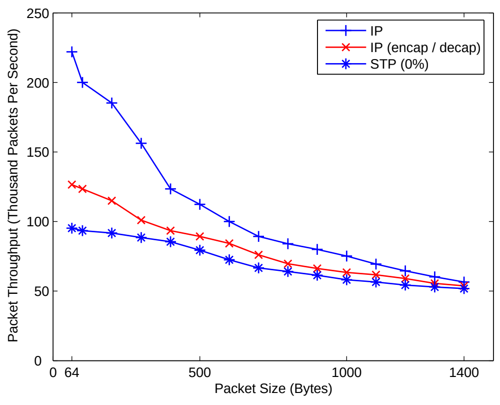


Figure 6: CPU Utilization of IP, IPsec, stealth probing sampling at 2%, and stealth probing with consistent hashing in the data throughput experiment.

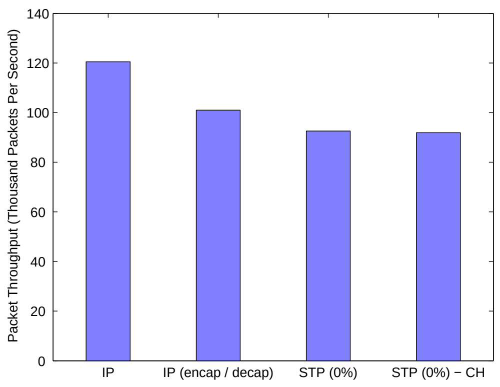
It can also be seen that the user-space PPM module is fairly non-intrusive for the processor since it contributes to only 1% of the total load in both cases. We thus believe that even in high-end routers performance monitoring could be the responsibility of the CPU, whereas stealth probing would have to be efficiently implemented in the data plane.

6.5 Packet Throughput

Finally, we compare the maximum zero-loss packet throughput of stealth probing at zero sampling rate (which is equivalent to that of IPsec), with and without consistent hashing, with that of IP. Fig. 7(a) plots the performance achieved for input traffic of fixed packet size, as a function of the packet size. It can be seen that the overhead introduced by stealth probing drops as packet size increases, becoming almost negligible for 1400-byte packets (which are used in the TCP-based throughput experiments of Section 6.3). The bad performance for small packet sizes is due to the large (fixed) overhead for processing packet headers, which is, however, more relevant for small packets and is gradually being amortized as packet size grows. The throughput



(a) Fixed Packet Size



(b) Mixed Input Traffic

Figure 7: Packet throughput (a) under fixed-size traffic and (b) under mixed traffic.

achieved for a more realistic traffic-mix is shown in Fig. 7(b). Packet sizes are drawn from a probability distribution that approximates traffic collected by NLANR during February 2001 [6, 7]. In this case, as expected, the impact of stealth probing becomes less pronounced, since the large processing overhead for small packets is compensated by the smaller overhead for large packets.

By isolating the processing steps in surelinks we observed that the IPsec/ESP encapsulation step is, somewhat unexpectedly, CPU intensive. This processing overhead translates to a significant impact on throughput as shown in Fig. 7(a) and Fig. 7(b) (see the data marked as "IP encap/decap," which where obtained after a dummy encapsulation and decapsulation step was added). We attribute this to the fact that, in the current Click implementation, encapsulation involves a dynamic memory allocation and a memory copy operation per packet. We believe that this overhead can be eliminated in a more optimized implementation.

7. RELATED WORK

Anticensorship systems such as Peacefire [4] and Psiphon [5] use encryption to avoid firewalls. Encryption has also been proposed to enforce *net neutrality* [38]. As argued earlier in this paper, encryption by itself is not sufficient to counteract discrimination. Perlman [33] proposed encryption between neighboring routers to make data and control traffic indistinguishable and per-packet acknowledgments to monitor the link between the routers. Surelinks use encryption between remote routers where availability monitoring is performed using implicit probing based on sampling, which makes monitoring unobtrusive.

Fatih [30] is a system for detecting and isolating malicious routers using a traffic validation method that relies on clock synchronization. Therefore, successful clock-synchronization attacks could compromise the system. Because Fatih does not specify a secure clock synchronization protocol, we cannot directly compare its security and efficiency. Stealth probing does not depend on clock synchronization. Furthermore, Fatih sends data packets in the clear, allowing an adversary to selectively target the traffic. Surelinks tunnel and encrypt the traffic to prevent this attack.

Secure traceroute [32] is a scheme for secure fault localization that could conceivably be applied at the path level. In secure traceroute, data packets between an *initiating* and a *responding* router are selected by the initiator to serve as probes by embedding in them secret identifiers. However, the responder stores replies for later retrieval [28], opening the possibility for delay attacks. Furthermore, data packets are neither encrypted nor tunneled, allowing an adversary to target individual components of the aggregate traffic.

Listen [35] and the data-plane monitor of the Feedback-Based Routing system [40] detect data-plane attacks by a combination of passive measurements of TCP traffic and insecure active probing. As such they have the limitations outlined in Section 4.1.

We previously introduced the idea of stealth probing in a short paper [11]. This paper makes the following additional contributions: First, we articulate in detail the design decisions such as why monitoring availability from edge routers is more preferable to monitoring at end hosts. Second, we refine the design and, for example, dismiss secure active probing, proposing instead to split traffic on multiple paths and do passive probing on each path. Third, we present a methodology for inferring path-performance using statistical sampling theory. Fourth, we present in detail a prototype implementation of the system and, finally, a thorough evaluation of the prototype in a testbed.

8. CONCLUSION

We have proposed *surelinks* as a primitive against discrimination that combines proactive and reactive countermeasures. The proactive countermeasure is an encrypted tunnel between a pair of nodes carrying a mix of traffic. The reactive countermeasures reroute traffic away from paths with poor performance using multipath routing and secure availability monitoring. Surelinks can be readily deployed in today's Internet for two important reasons. First, surelinks are *backward compatible* with the existing infrastructure since IP tunnels can be deployed across legacy routers and networks. Furthermore, commercial routers increasingly offer IP tunneling and encryption at line rates. Second, any pair of communicating parties can deploy surelinks irrespective of the participation of external parties, making surelinks *incrementally deployable*. As such, surelinks offer immediate benefits even during limited deployment.

In the future, we plan to study algorithms for spreading traffic among multiple paths in a surelink, or multiple surelinks, in a coordinated fashion. We also plan to explore using surelinks to provide secure interdomain communication within small groups of participating networks [12].

9. REFERENCES

[1] *Towards a Two-tier Internet*, Dec. 2005. http://news.bbc.co.uk/1/hi/technology/4552138.stm.

[2] *Net Neutrality Amendment Rejected*, Jun. 2006. http://www.washingtonpost.com/wp-dyn/content/article/2006/06/28/AR20060%62802176.html.

[3] Keynote launches new SLA services, Jun. 2001. http://investor.keynote.com/phoenix.zhtml?c=78522&p=irol-newsArticle_Pr%int&ID=183745&highlight=.

[4] *http://www.peacefire.org/*.

[5] *http://psiphon.civisec.org/*.

[6] *http://advanced.comms.agilent.com/*.

[7] *http://pma.nlanr.net/Datacube/*.

[8] *http://www.xorp.org/*.

[9] *Meraki Networks, http://meraki.net/*.

[10] Andersen, D., Balakrishnan, H., Kaashoek, F., and Morris, R. Resilent overlay networks. In *Proc. of ACM Symposium on Operating System Principles* (Oct. 2001).

[11] Avramopoulos, I., and Rexford, J. Stealth probing: Efficient data-plane security for IP routing. In *Proc. USENIX Annual Technical Conference* (May/Jun. 2006).

[12] Avramopoulos, I., and Rexford, J. A pluralist approach to interdomain communication security. In *Proc. ACM Workshop on the Economics of Networked Systems and Incentive-Based Computing* (Jun. 2007).

[13] Avramopoulos, I., Syrivelis, D., Rexford, J., and Lalis, S. Secure availability monitoring using stealth probes. Technical Report TR-769-06, Dept. of Computer Science, Princeton University, Oct. 2006.

[14] Bacceli, F., Machiraju, S., Veitch, D., and Bolot, J. The role of PASTA in network measurement. In *Proc. ACM SIGCOMM* (Sept. 2006).

[15] Dierks, T., and Allen, C. The TLS protocol version 1.0. RFC 2246, IETF, Jan. 1999.

[16] Duffield, N., and Grossglauser, M. Trajectory sampling for direct traffic observation. *IEEE/ACM Trans. Networking 9*, 3 (Jun. 2001), 280–292.

[17] Felten, E. Nuts and bolts of network neutrality, Jul. 2006. http://itpolicy.princeton.edu/pub/neutrality.pdf.

[18] Goldenberg, D., Qiu, L., Xie, H., Yang, Y. R., and Zhang, Y. Optimizing cost and performance for multihoming. In *Proc. ACM SIGCOMM* (Aug./Sept. 2004).

[19] Guo, F., Chen, J., Li, W., and Cker, T. Experiences in building a multihoming load balancing system. In *Proc. IEEE Infocom* (Mar. 2004).

[20] Harkins, D., and Carrel, D. The Internet Key Exchange (IKE). RFC 2409, IETF, Nov. 1998.

[21] Hopps, C. Analysis of an equal-cost multi-path algorithm. RFC 2992, IETF, Nov. 2000.

[22] Karger, D., Lehman, E., Leighton, T., Levine, M., Lewin, D., and Panigrahy, R. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *Proc. ACM STOC* (May 1997).

[23] Kent, S., and Atkinson, R. IP Encapsulating Security Payload (ESP). RFC 2406, IETF, Nov. 1998.

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

[25] Kohler, E., Morris, R., Chen, B., Janotti, J., and Kaashoek, F. The Click modular router. *ACM Transactions on Computer Systems 18*, 3 (Aug. 2000), 263–297.

[26] Krawczyk, H. The order of encryption and authentication for protecting communications (or: How secure is ssl?). In *Proc. CRYPTO* (Aug. 2001).

[27] Lehmann, E. L. *Testing Statistical Hypotheses*. Wiley, New York, 1959.

[28] Mathur, G., Padmanabhan, V., and Simon, D. Securing routing in open networks using secure traceroute. Technical Report MSR-TR-2004-66, Microsoft Research, Jul. 2004.

[29] Mcpherson, D., and Labovitz, C. Worldwide infrastructure security report, Volume II. Tech. rep., Arbor Networks, Sept. 2006.

[30] Mizrak, A., Cheng, Y.-C., Marzullo, K., and Savage, S. Fatih: Detecting and isolating malicious routers. In *Proc. International Conference on Dependable Systems and Networks* (Jun. 2005).

[31] Nucci, A. Skype detection: Traffic classification in the dark, Jul. 2006. http://www.narus.com/_pdf/news/Converge-Skype%20Detection.pdf.

[32] Padmanabhan, V., and Simon, D. Secure traceroute to detect faulty or malicious routing. In *Proc. ACM SIGCOMM HotNets Workshop* (Oct. 2002).

[33] Perlman, R. *Network Layer Protocols with Byzantine Robustness*. PhD thesis, Massachusetts Institute of Technology, Aug. 1988.

[34] Prasad, R., Dovrolis, C., Murray, M., and Claffy, K. Bandwidth estimation: Metrics, measurement techniques, and tools. *IEEE Network* (Nov.-Dec. 2003).

[35] Subramanian, L., Roth, V., Stoica, I., Shenker, S., and Katz, R. Listen and Whisper: Security mechanisms for BGP. In *Proc. Symposium on Networked System Design and Implementation* (Mar. 2004).

[36] Thompson, S. *Sampling*, second ed. Wiley Series in Probability and Statistics. Wiley, 2002.

[37] Wendlandt, D., Avramopoulos, I., Andersen, D., and Rexford, J. Don't secure routing protocols, secure data delivery. In *Proc. ACM SIGCOMM HotNets Workshop* (Nov. 2006).

[38] Yang, X., Tsudik, G., and Liu, X. A technical approach to net neutrality. In *Proc. ACM SIGCOMM HotNets Workshop* (Nov. 2006).

[39] Yuan, R., and Strayer, W. *Virtual Private Networks: Technologies and Solutions*. Addison Wesley, 2001.

[40] Zhu, D., Gritter, M., and Cheriton, D. Feedback based routing. In *Proc. ACM SIGCOMM HotNets Workshop* (Oct. 2002).