On Network Neutrality Measurements

ALEX MALTINSKY and RAN GILADI, Ben-Gurion University of the Negev, Israel
YUVAL SHAVITT, Tel-Aviv University, Israel

Network level surveillance, censorship, and various man-in-the-middle attacks target only specific types of network traffic (e.g., HTTP, HTTPS, VoIP, or Email). Therefore, packets of these types will likely receive "special" treatment by a transit network or a man-in-the-middle attacker. A transit Internet Service Provider (ISP) or an attacker may pass the targeted traffic through special software or equipment to gather data or perform an attack. This creates a measurable difference between the performance of the targeted traffic versus the general case. In networking terms, it violates the principle of "network neutrality," which states that all traffic should be treated equally. Many techniques were designed to detect network neutrality violations, and some have naturally suggested using them to detect surveillance and censorship. In this article, we show that the existing network neutrality measurement techniques can be easily detected and therefore circumvented. We then briefly propose a new approach to overcome the drawbacks of current measurement techniques.

CCS Concepts: • **Networks → Network control algorithms**; **Network measurement;**

Additional Key Words and Phrases: Adversarial model

ACM Reference Format:
Alex Maltinsky, Ran Giladi, and Yuval Shavitt. 2017. On network neutrality measurements. ACM Trans. Intell. Syst. Technol. 8, 4, Article 56 (May 2017), 22 pages.
DOI: http://dx.doi.org/10.1145/3040966

1. INTRODUCTION

A major security hazard is the possibility to target specific sensitive traffic and treat it differently. For example, a foreign government may divert all Voice Over IP (VoIP) traffic traversing a router and pass it through data gathering equipment to collect sensitive information [Zhang et al. 2009b]; a nondemocratic government may divert all traffic destined to specific websites in order to block antigovernment activities [Sfakianakis et al. 2011; Gill et al. 2015]; or a criminal group may hijack traffic destined to a financial institution in order to penetrate user accounts [Zhang et al. 2010; Cunha et al. 2014]. This type of discriminatory activity creates a measurable difference between the performance of the targeted traffic versus the general case. In networking terms, it violates the principle of network neutrality, which states that all traffic should be treated equally.

The term "Network Neutrality" was coined by Tim Wu outside of the security domain in a paper discussing the regulation of competition between products on the privately owned infrastructure of the Internet [Wu 2003]. Network neutrality is the notion that

This work is supported by a grant from the Blavatnik Interdisciplinary Cyber Research Center (ICRC) at Tel Aviv University.

Authors' addresses: A. Maltinsky and R. Giladi, Communication Systems Engineering, Ben-Gurion University of the Negev, Beer-Sheba, Israel; emails: maltalex@gmail.com, ran@bgu.ac.il; Y. Shavitt, School of Electrical Engineering, Tel-Aviv University, Ramat Aviv 69978, Israel; email: shavitt@eng.tau.ac.il.

DOI: http://dx.doi.org/10.1145/3040966

networks should behave as neutral carriers of information, and should not discriminate between data of different applications, of different content, users, destinations, etc. Nonneutral network behavior can appear for purposes outside the security domain: carriers may slow down traffic of services that competes with their own, or nontype of applications that consumes large traffic volumes, such as peer-to-peer traffic.

There is a need for a tool that can detect network neutrality violation in order to warn users that they are under surveillance. Indeed, the networking community has developed several such tools with the aim of detecting network neutrality violations [Dischinger et al. 2008, 2010; Zhang et al. 2009a; Kanuparthy and Dovrolis 2010; Tariq et al. 2009; Lu et al. 2007, 2010; Electronic Frontier Foundation (EFF)]. However, these tools were developed with disregard to the fact that a spying nation or a group of criminals will invest efforts to conceal their activities. Thus, when building a network neutrality measurement tool one must conceal its measurement activities.

Generally speaking, measurements and measurement tools have two important attributes associated with them: accuracy and precision. Precision, as defined by the International Organization for Standardization (ISO), is "the closeness of agreement between independent test/measurement results obtained under stipulated conditions," that is, the degree with which repeated measurements of the same phenomena produce the same results [International Organization for Standardization 2006]. Accuracy, on the other hand, is defined as "closeness of agreement between a test result or measurement result and the true value" [International Organization for Standardization 2006]. Naturally, measurements can exhibit both features, one of them, or none at all.

This work focuses on an underlying assumption, which is common to all existing network neutrality measurement techniques, that the network being measured is naive, and it does not attempt to affect the results of the neutrality measurement. We show that due to this assumption, the measurements performed by existing tools are susceptive to manipulations. Namely, the measurements may be precise but their accuracy is questionable as they overlook a significant source of possible systematic errors that can completely skew the results.

We show here, using real Internet traces, that neutrality measurements as suggested by all previous work can be easily detected and thus are susceptive to manipulation by the entity that desires to conceal its nonneutral behavior (Section 4). We also show using a testbed emulation that such manipulation will not significantly affect other traffic in the network (Section 5). Finally, we suggest a novel design for a stealth neutrality measurement tool (Section 6).

2. RELATED WORK

Previous network neutrality tools treated the ISPs, or *the network*, as the entity that attempts to perform nonneutral treatment, thus from now on we will use here these terms. Specifically, we will assume that the traffic discrimination is performed by the ISP.

Generally speaking, one can divide network neutrality measurement techniques into four categories by distinguishing between active and passive techniques, and by distinguishing between unilateral and bilateral techniques. The vast majority of network neutrality measurement tools use active measurements. Namely, they send special measurement traffic and use it to evaluate the characteristics of the network in question. Passive measurement techniques, on the other hand, are those techniques that make their measurements passively without sending any special traffic. Unilateral measurement techniques are those that a single host can perform without the need for a second party, whether it be a host or a special measurement server. Bilateral tools, in contrast, are those that require the participation of two hosts or a host and a server. This work shows that current tools can be easily detected by the ISPs and thus can be manipulated. Next, we will describe the most common tools used today (or recently suggested), and examine their vulnerabilities.

Glasnost [Dischinger et al. 2008, 2010], which is an active bilateral tool, enables end users to check the neutrality of their ISP, by measuring network performance in relation to different transport layer port numbers and application layer signatures. It gives users immediate feedback regarding their connection, but also aggregates results to reach conclusions about the policies of an entire ISP. A Java applet downloaded by the user performs the measurement to one of a few Glasnost servers.

NetPolice [Zhang et al. 2009a], which is an active unilateral tool, is designed to detect differentiation in core networks. It targets a specific Autonomous System (AS) and analyzes it. NetPolice attempts to measure not only application based differentiation, but also differentiation based on the AS from which the traffic comes (previous-AS) and the AS where the traffic is sent to (next-AS). NetPolice employs a probing technique, similar to traceroute: periodically sending packets with increased TTL (Time to Live) values. However, unlike traceroute, the packets are large, have transport layer headers with port numbers of known applications, and their payload imitates the application content. Since dropped probe packets will not trigger ICMP reply, NetPolice can estimate the loss rate of various applications to any node along a path. The system, deployed on PlantLab, selects paths to cover as many ingress-egress pairs of an AS. In each path, the loss rate inside the target AS is calculated as the difference in loss rate between the egress and ingress nodes of that AS. By comparing the loss rate distribution of different applications along the same path to the loss rate distribution of HTTP, which is taken as a baseline, NetPolice identifies which applications receive poor treatment and therefore more packet loss.

Diffprobe [Kanuparthy and Dovrolis 2010], which is an active bilateral tool, aims at identifying discriminatory scheduling and active queue management algorithms deployed in a provider's network. Diffprobe assumes that neutral networks employ First-Come First-Served (FCFS) scheduling with drop-tail queue management; nonneutral networks will classify traffic and prioritize some flows at the expense of others. This assumption means that the target network will treat flows differently only during periods of high load. Diffprobe is running its test traffic between two computers in a client-server configuration. The system simultaneously injects the network with two time-stamped flows: an application flow, assumed to be classified as low priority, and a neutral probing flow. The receiver records packet loss and delay.

NANO [Tariq et al. 2009] views the network as a "black box" that gives users various types of services. It then relies on passive unilateral performance measurements of these services to estimate a network's neutrality. This approach has the advantage of being completely independent of any specific differentiation mechanism. NANO uses passive performance measurements by the NANO agent deployed on volunteers' computers. The agent passively listens to all traffic going through the user's machine and collects various performance measurements such as throughput and delay of flows it detects. It also collects information about the computer and its environment such as its location, type of connection, operating systems, and hardware specification. Information from all NANO agents is sent to a central NANO server where it is stored and analyzed.

POPI (Packet fOrwarding Priority Inference) [Lu et al. 2007, 2010] is a bilateral active technique designed to infer packet forwarding priorities in routers by saturating the bottleneck link along the path. A sender transmits n_b packet bursts (well spaced), each carrying n_r packets of each of the k packet types being tested, giving a total of $n_r k$ packets per burst. The receiver calculates for each packet type i the Average Normalized Rank (ANR) using $ANR_i = \frac{1}{n_b}\sum_{m=1}^{n_b}\frac{r_i^m}{k}$ where r_i^m is the rank of traffic type i in burst m. The ANR values are then clustered to classes of equal performance.

Switzerland, whose latest design document (version 0.3) dates back to 2008, is a passive bilateral technique [Electronic Frontier Foundation (EFF)]. Switzerland looks for lost, modified, and injected packets in a flow between a pair of hosts. It does so using a central server to which both hosts report the cryptographic hashes of sent and received packets along with the time when the packets were observed. The server then looks at the packets in search of anomalies. It searches for packets that were received with a hash different from the hash reported during transmission, which indicates modified packets. It also looks for packets injected by the network, that is, packets that the receiver reports but the sender does not. Lastly, it counts dropped packets by comparing the number of packets observed by the receiver and comparing it to the number of packets sent. The design of Switzerland is not complete and the project seems to have been abandoned.

3. MODELING ISP BEHAVIOR

As shown in Section 2, there are quite a few different techniques for determining whether a network is biased against certain kinds of traffic or users. These techniques measure metrics such as throughput, delay, and packet loss to statistically evaluate the behavior of a network. However, all these techniques assume a *naive* network operator, which, as discussed in this section, may not be true in practice and may lead to false negative results.

Although it is desirable to have a single yardstick to grade how the different measurement techniques perform against a sophisticated ISP, which attempts to obscure its operation, their versatility makes this task challenging. Instead, we define a model of a sophisticated ISP behavior and show how different network neutrality measurement can be manipulated to skew their results.

3.1. The Naive Network Operator

The existing measurement techniques, mentioned in the previous section, assume that the network they are measuring is naive. This means that those techniques assume the network does not actively interfere with the measurement and that all inaccuracies are only due to noise. However, it is easy to imagine ISPs leveraging the knowledge of and power over the traffic flowing through their networks to manipulate these well-known techniques. Such manipulation can be realized as either complete blockage of these measurement techniques or, more subtly, by prioritizing measurement traffic or servers to guarantee a favorable outcome. This way, nonneutral ISPs can reduce the chance of detection. This fact should be taken into account when evaluating traffic differentiation measurement techniques.

3.2. The Defensive ISP Model

The previous discussion leads to a new model for the networks being measured. In the **defensive ISP model**, ISPs proactively defend their practices and employ reasonable means aimed at obfuscating their policies. Specifically, two key assumptions are made about the network operators: they are both *omnipotent* and *omniscient*.

The omniscient assumption states that the network operator is familiar with all network neutrality measurement techniques. This notion has its roots in Kerckhoffs's principle, which states that the "adversary always knows the system." Applied mostly in the area of security, this principle makes sense in our context, as defensive ISPs can follow the research in this area. However, if the measurement uses some randomness, the defensive ISP cannot know the random values in advance. Another sense in which the network operator can be considered omniscient is that it has full access to the traffic flowing through its network and it can read and analyze it.

The ISP is omnipotent in the sense that it has full control over user traffic. It can modify, drop, and delay traffic as well as inject any type of packet into a user flow. The network operator is only limited by two factors: the perceived experience of its users and the computational resources it needs to perform an action. It cannot perform actions that affect the everyday experience of a significant portion of its users, nor can it perform actions that are extremely resource intensive, such as deciphering the contents of encrypted traffic without knowing the encryption key.

Thus, the defensive ISP will use all its power to identify measurement traffic in its network, and then change this traffic treatment to skew the measurement result, or alternatively, drop this traffic to block the measurement. This means that the question of the susceptibility of different measurement techniques when faced with a defensive ISP becomes first and foremost one of classification. Once a measurement technique has been identified by this omnipotent entity, little can be done to ensure the technique accuracy.

3.3. Traffic Authenticity

Besides the challenge of remaining undetected, the defensive ISP model brings a second significant problem to network neutrality measurement. The traditional practice in the field states that network performance metrics to be measured are based on throughput, delay, and packet loss. However, a defensive ISP that, as stated earlier, is both omniscient and omnipotent has the capability of limiting network usage without directly affecting these metrics.

Consider, for example, an ISP interested in limiting Bit-Torrent file transfers. Such an ISP could modify legitimate Bit-Torrent control messages to make peers see fewer sources to download from. This packet modification could be used at the edges of a network to reduce the number of uploads from an ISP's customers to other networks, saving it money on bandwidth costs. Such intervention will not be visible at all by techniques measuring only packet loss or delay metrics. Techniques that measure throughput might miss it as well, as the sessions that are established can achieve a normal rate. Only by examining the aggregate throughput of many users or by checking packets for authenticity can one detect this intervention. This creates a need for a network neutrality measurement technique that is capable of dealing with such a threat, adding packet authenticity as a new metric that network neutrality tools should measure.

4. NETWORK NEUTRALITY MEASUREMENT DETECTION

4.1. How to Detect a Neutrality Measurement

In a response to the Canadian Radio-television and Telecommunications Commission as part of the "review of the Internet traffic management practices of Internet service providers," Sandvine, the company whose equipment was used by Comcast to send the TCP RST packets in 2007, said it used two primary types of techniques for packet classification: a behavioral flow-based method and a signature-based method [SAN 2008]. Assuming that this is the industry standard for packet classification, what has to be analyzed for detectability is both the general behavior exhibited by network neutrality measurement techniques as well as the specific signatures, such as certain strings or patterns appearing in its packets. Sections 4.1.1 and 4.1.2 deal with behavioral classification of popular measurement approaches, while Section 4.1.4 deals with signature-based approaches. In the discussion of behavioral classification we assume that measurement techniques that exhibit a distinct behavior are easier to detect than those whose behavior resembles regular traffic, since for the latter a high rate of false positives is expected.

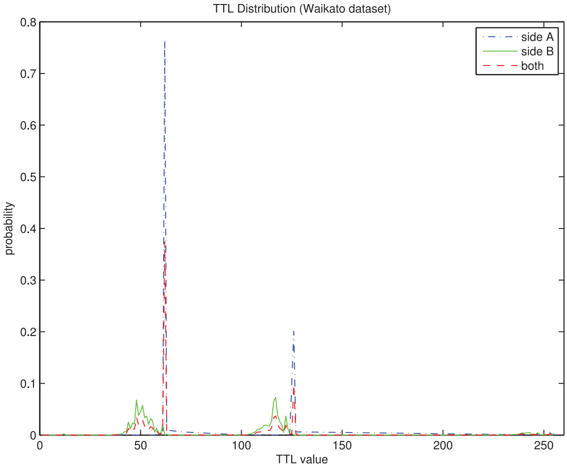

Fig. 1. Waikato TTL distribution.

4.1.1. Behavioral Detection of NetPolice. Identifying NetPolice probing packets, from an ISP's point of view, is fairly easy since all modern operating systems use well-known constant TTL values when sending IP traffic [Default TTL 2009]. Modern versions of Windows use a value of 128; Linux and Linux-based operating systems use a value of 64. A value of 255 is used by the Solaris operating system and in ICMP packets on some Linux kernels. This fact guarantees that routers, especially access routers, see very specific TLL values when dealing with regular traffic. Based on that, ISPs can assume that all packets with abnormal TTL values (that are not slightly lower than 64, 128, or 255) belong to some type of measurement technique and therefore should be treated differently than other packets.

To verify this point, we analyzed a 4-day traffic trace (1–5 of August 2005) from the WAND group (part of the "Waikato I" trace [WAND Group 2010]). This trace was recorded on an access link of the University of Waikato in New Zealand to its Internet provider. It contains over 165GB of traffic and almost 500 million packets. The distribution of TTL values in IP packets in this trace is plotted in Figure 1.

Even these basic results show several distinct peaks in the TTL values on this access link, corresponding to the values close to, but slightly lower than, the known TTL constants. If separated by source MAC addresses into "Side A" and "Side B," and therefore by the packet direction on the link, the results become even clearer, as shown in Figure 1.

The distribution of TTL values on side A has two distinctive spikes on TTL values of 62 and 126. In fact, over 96% of the 241 million packets in this direction are concentrated in these two spikes. Side B's distribution is not as concentrated, but still has two large spikes around 50 and 117 as well as a third, less significant spike around 247. This distribution is considerably less concentrated compared to the other direction: but still it covers a small range of TTL values, which are slightly shifted to lower values of the typical 64, 128, or 255. On this side, 95.5% of the 251 million packets are located in one of these typical ranges: 41–61, 105–124, or 231–235. It is therefore clear that side A is the uplink direction from the university and that the point where the TTL measurement took place is located two hops from the majority of university computers. Side B, on the other hand, is the downlink from the Internet to the university. It still has spikes caused by the typical TTL values, but the spikes are shifted to lower values and are not as sharp. This is due to the different distances at which the external end

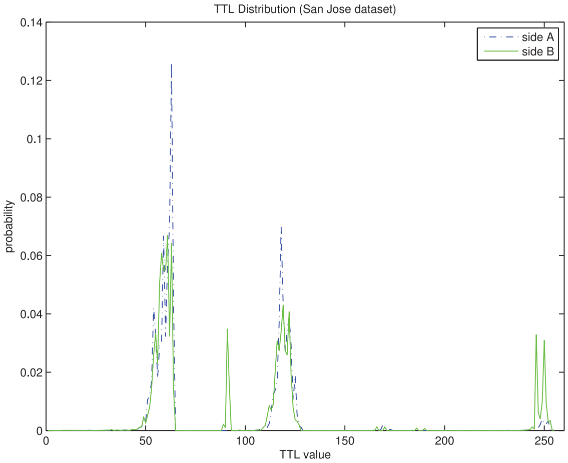

Fig. 2. San Jose TTL distribution.

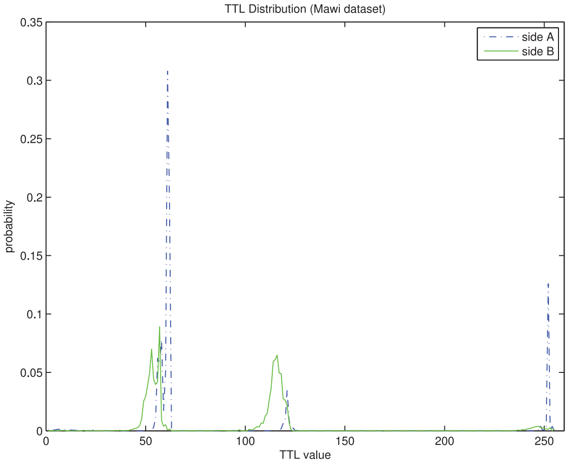

Fig. 3. Mawi TTL distribution.

points are located, about 5 to 20 hops away from the university. Note that these traces are anonymized and the IP addresses of these packets are encrypted. Therefore, it is impossible to use the IP addresses to establish the flow direction.

To examine the TTL distribution in core links, we used two datasets. Figure 2 depicts the distribution of TTL values of an 11 minute trace taken from the Equinix datacenter in San Jose, CA, captured by CAIDA [2011]. This trace contains 675 million packets from 13:00 to 13:11 UTC, April 13th, 2011, from both directions, on a 10Gbps link. Figure 3 depicts the distribution of TTL values of a 4 hour trace taken on April 15, 2010 from a 150Mbps trans-Pacific link by the MAWI working group [MAWI working group 2010].

The TTL distributions in core links, plotted in Figures 2 and 3 show a similar distribution to that of the Waikato access link. The vast majority of packets lie in very specific ranges, indicating it is quite easy to identify packets belonging to TTL-based measurement techniques, even if the detection point is a core link. The single anomaly is the spike around the value 91 on side B of the San Jose trace in Figure 2. Investigating

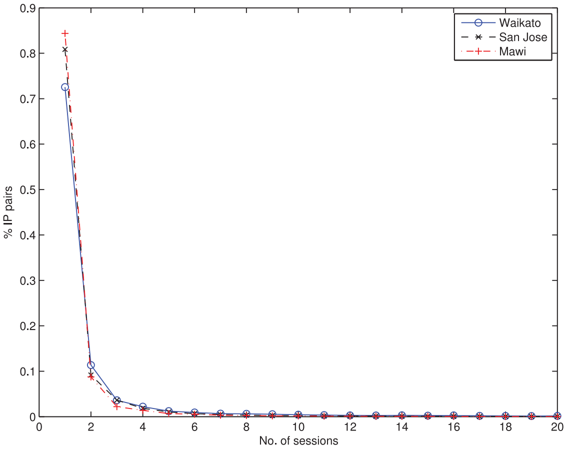

Fig. 4. The distribution of the number of sessions for an IP pair (interval concurrent).

the packets with this TTL value showed that the overwhelming majority of this traffic comes from a few discrete IP addresses. Since the IP addresses in the trace are anonymized, it is impossible to determine the networks these IPs belong to. However, the anonymization in all traces is prefix preserving. Thus, it is possible to tell that the IPs that send considerable quantities of packets with this uncommon TTL value originate from two distinct /24 subnets. These subnets are probably either very far away in terms of hop count, are poorly configured, or perhaps use a custom TTL value for some reason.

4.1.2. Behavioral Detection of the Flow Pair Approach. A more popular neutrality measurement technique is to compare the performance of traffic flows belonging to different applications by creating several concurrent or back to back flows between two hosts. This approach is used by Glasnost [Dischinger et al. 2010], POPI [Lu et al. 2010], and Diffprobe [Kanuparthy and Dovrolis 2010].

To check for detectability, we examined, using the traffic traces that were analyzed previously, how common it is for a pair of hosts to open several connections concurrently. Figures 4 and 5 show the distribution of the number of "concurrent" sessions between pairs of hosts. The figures differ in the definition of concurrent: in Figure 5 we require two concurrent sessions to have at least three packets with sampling times $t_1 < t_2 < t_3$ such that the packet associated with t_2 belongs to one session and the other two packets belong to the other session (termed true concurrent); in Figure 4 we require the two sessions to be in the same 15 minute time window (termed interval concurrent).

A session was defined as either a four-tuple consisting of a pair of IP addresses and UDP/TCP ports in the case of UDP and TCP flows, or just a pair of IPs and a type of protocol for other protocols sent over IP such as ICMP. Note that TCP flows that contain only unacknowledged SYN packets were not counted as they do not represent actual sessions but rather unsuccessful handshake attempts often created by port scanning. TCP packets with the RST flag on were also ignored for the same reason.

The three datasets are taken from different trunk speeds, and thus a 15 minute interval may represent an uneven number of IP pairs for the statistics. Thus, we randomly chose a 15 minute interval from the San Jose dataset; it contained almost 934 million packets forming a session connecting about 15.25 million unique IP pairs. We then took multiple 15 minute intervals of the other datasets until we roughly got the same number of IP packets. (For Mawi we took 35 intervals with 930 million packets

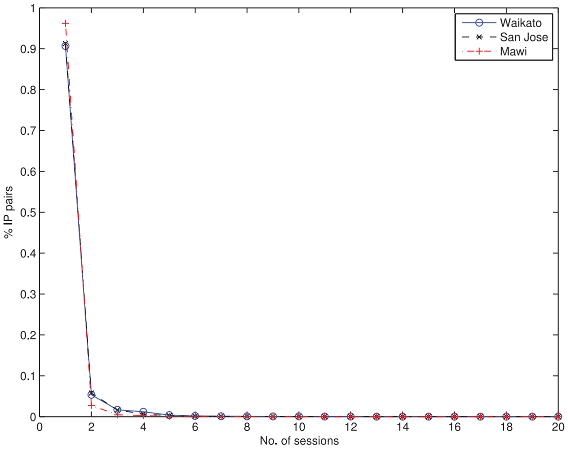

Fig. 5. The distribution of the number of sessions for an IP pair (true concurrent).

and 22.9 million unique pairs, and for Waikato we took 768 intervals with 975 million packets and 13.6 million pairs.[1])

For the true concurrent definition we found that 90%–96% of the IP pairs had no concurrent sessions; for the interval concurrent the number is a bit lower, but still fairly high 72%–84% of the pairs. Namely, at least 3/4 of the IP pairs are not suspicious of measurement activity. We also checked the sensitivity of this result to the interval length and found stability of the percentage of single session IP pairs: for the SJ dataset the number of IP pairs with only a single true concurrent session is 91.26%, 90.81%, and 90.86% for intervals of 15, 8, and 4 minutes, respectively; for interval concurrency these percentages are 80.86%, 81.14%, and 82.32%.

To summarize, the basic approach that many network neutrality measurement techniques use creates traffic that in its behavior differs greatly from the behavior of regular Internet traffic. Since ISPs can use relatively simple hardware at wire speed to keep track of the number of concurrent sessions of its clients, ISPs can easily flag users with an abnormal amount of concurrent connections to the same destination. After identifying suspicious clients, the little left flagged traffic is manageable; it can all be given the same priority, causing measurements to come out as neutral. Alternatively, flagged traffic can be sent for further analysis to more accurate, possibly signature-based, equipment which now has to work at only a fraction of the original line rate.

4.1.3. Discussion of IP Spoofing. The conclusion from Section 4.1.2 calls for a seemingly trivial workaround to avoid measurement traffic detection and classification, that is, use source IP spoofing to have some flows sent from one node, using different IP addresses and still reach the same destination. By using this idea, one can augment the design of the current approach to avoid detection. Instead of using two or more flows between a pair of IP hosts (a single source and destination), which is uncommon and detectable, one can use only one flow between the real IP addresses of both parties. That flow will be used for parameter negotiation. The actual test traffic will be sent from spoofed sources, and the results sent back using the real flow. This way, the ISP will see only a single session between each pair of IP hosts, circumventing the problem presented in Section 4.1.2. One should note, though, that using different IP source

[1] We summed the unique IP pairs per interval, thus an IP pair that appears in more than one interval is counted multiple times.

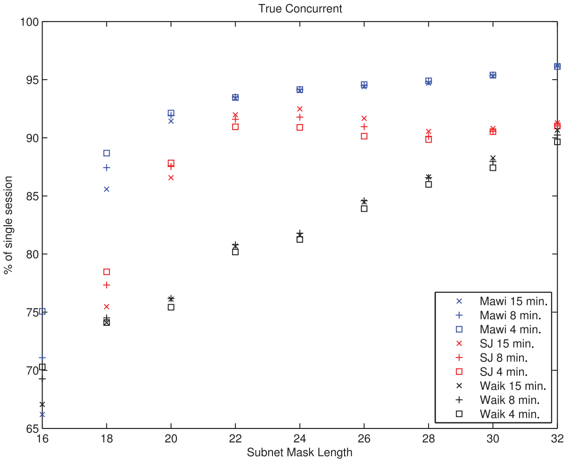

Fig. 6. The portion of subnet pairs that show only a single true concurrent session as a function of the subnet size.

addresses may cause two flows to be routed differently in load balancers, which might result in large false positives for network neutrality tests.

However, spoofing is not easy and requires software installation with root permission; a Glasnost style applet cannot send spoofed addresses. Difficulties due to the network are even more severe. First, there are existing tools such as reverse path forwarding filters that network operators use to find and drop packets with spoofed IP addresses. These tools are quite common [Beverly et al. 2009] and their use is encouraged due to security concerns. Moreover, last mile technologies such as DSL and cable DOCSIS modems have a built-in ability of binding IP addresses to link layer addresses, thus greatly limiting the availability of IP spoofing where it is needed the most—homes and small business. The MIT Spoofer project reported that only an estimated 11.1%±4.1% of IPv4 addresses are able to send a packet with a fake private, unallocated, or a valid (according to Border Gateway Protocol (BGP) tables) address [Beverly et al. 2009]. Among the clients unable to do any of these types of spoofing, approximately 77% can still spoof any address within their /24 subnet and 44% can spoof within their /20 subnet.

Although the ability to send traffic with a fake source IP address is not ubiquitous, a significant portion of hosts can still send packets with source IP addresses in their own /24 to /20 subnets. These users can open several sessions by using different source IP addresses within their own subnets. From the ISP point of view, this will appear as a number of different communicating IP pairs. This raises the question of how common it is for hosts that belong to two subnets, to be engaged with more than one concurrent session with each other. The more common this behavior is, the harder it will be for a defensive ISP to detect measurement techniques using this method (i.e., subnets spoofing).

We repeated the analysis of Figure 5 with different masks to test how common are true concurrent sessions between subnets of various sizes. For each subnet pair we only counted the maximal number of concurrent sessions. Figure 6 depicts the results for 15, 8, and 4 minute intervals from the Mawi and San Jose (SJ), and Waikato datasets (note that /32 is the case of no spoofing). For the SJ and Mawi traces, there is clearly no significant difference in the number of pairs with a single concurrent session for subnets in the range between /24 (the maximal "widely" possible spoofing subnet range) and /32 (no spoofing), and only a slight drop for /20, which are somewhat possible for

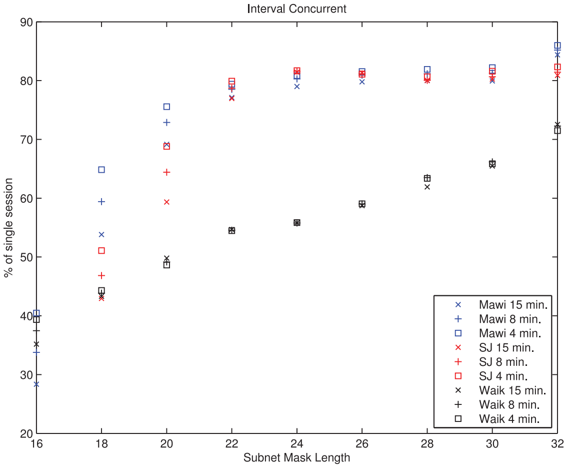

Fig. 7. The portion of subnet pairs that show only a single interval concurrent session as a function of the subnet size.

spoofing. Similar results were obtained for interval concurrent sessions (Figure 7). This suggests that spoofing can help in avoiding detectability only if one can spoof an IP address outside its /20 subnet, which is possible, but not very common. For the vast majority of Internet users, however, such IP spoofing cannot be used to hide the concurrent sessions created by measurement techniques. The Waikato trace implies better results for spoofing since it is taken from the border of a university network, which is not representative of the commercial Internet.

The conclusion from these results is that even if spoofing up to /20 is assumed to be fairly common, the analysis shows that several concurrent sessions between two /20 subnets is not very likely in normal traffic. At a /20 mask, 8%–40% of subnet pairs had more than a single concurrent session. At a /24 mask, which according to the Spoofer project is a fairly common spoofing limit, this number drops to 6%–21% of subnet pairs.

4.1.4. Signature-Based Detection. Signature-based detection relies on specific strings or patterns that occur in certain types of traffic significantly more than in the general traffic. These patterns include constant or easily predictable strings and numeric values in packets, as well as known sequences of packets. Analyzing measurement techniques for particular signatures has to be done individually per technique, as they vary considerably. We only discuss the two open source techniques.

The easiest way to classify Glasnost measurements [Dischinger et al. 2010] is by looking for traffic to or from one of the Glasnost measurement servers. The fixed servers are hard coded into Glasnost source code, which is freely available. Even without the source code, it would have been easy to find the server addresses by simply running several measurements and observing the tool behavior.

POPI can be easily identified by one of several signatures. First, in its TCP-based control channel, POPI passes some constant strings as well as the parameters of the measurement in a fixed pattern. Second, all three types of traffic that POPI generates (TCP, UDP, and ICMP) have very distinct characteristics. Its ICMP traffic is malformed as it uses a value of 52 in its ICMP type field, a reserved value. The TCP traffic does not contain a TCP handshake or any means of closing a TCP session. All TCP packets have a nonzero acknowledgment number while the ACK flag is not raised, making the packets malformed. Moreover, many fields in the TCP header, including the TCP sequence number, are constant across all packets. Lastly, the payload of both TCP and

UDP packets contains many constant fields, easily predictable incremental counters, and a large amount of zero padding.

4.2. Passive Measurement Techniques

The only passive neutrality measurement techniques described in the literature are NANO [Tariq et al. 2009] and the abandoned Electronic Frontier Foundation's Switzerland project [Electronic Frontier Foundation (EFF)]. The following discussion focuses mostly on NANO, as its design is final and well documented, unlike Switzerland.

Passive measurement techniques have a great advantage in the area of network neutrality measurement since they avoid the "arms race" between measurement techniques and network policy enforcement, by generating nondetectable traffic. The only exception to this in NANO is the traffic that a NANO agent generates when reporting its observations to the central NANO server. This is not measurement traffic, but ISPs might treat users who were observed contacting known NANO servers differently. This issue was never addressed by NANO, possibly due to two reasons: First, such lists of users, who have accessed a certain server, are not easy to manage. Second, this problem is not hard to solve as long as the size of the reports is small. Instead of collecting the reports directly at the server, they can be collected through various "middlemen," such as file hosting services, proxies, and even webmail. For example, one could report to the main server by logging into Gmail or a similar service using HTTPS and emailing the results. Since the entire connection is encrypted and authenticated, the ISP has no way of knowing the mail destination or content. Above all, it is clear that treating all hosts who access a webmail service using HTTPS as potential NANO measurement points is far from practical as it will generate an unacceptable number of false positives.

However, passive measurements are rather limited in their abilities. For example, the only delay metric measured by NANO is the RTT between SYN and SYN/ACK, or between SYN/ACK and ACK packets in the TCP handshake, since these are the only packets in the TCP protocol that are practically guaranteed not to be delayed, for example, by the application at the other end of the connection. The RTT during the rest of the TCP flow as well as RTTs of any non-TCP flow are not measured, as they would not be reliable. Loss rates can be calculated only for TCP sessions, and only based on sequence numbers or retransmissions. Throughput measurements of observed flows might also give inaccurate readings as they heavily depend on the application that generates the traffic and its configuration. Checking traffic for authenticity is also tricky as it requires the exchange of information between participating hosts. Switzerland solves the authenticity problem by using a central server that collects cryptographic signatures of all sent and received packets; however, it is hard to see how this approach scales as hosts send information regarding each and every packet, thus sending large amounts of data in total, which is hard to hide. All of this adds a lot of uncertainty to the results passive techniques produce.

5. A PROOF OF CONCEPT

5.1. Overview and Design

To illustrate how easy it is for an ISP to skew the results of a network neutrality measurement tool using the previously discussed technique, we conducted a proof of concept experiment that demonstrates the practicality of such an intervention. We created a dedicated testbed to emulate a network neutrality measurement between two hosts connected through a nonneutral network. The nonneutral network employs various QoS mechanisms that treat some types of traffic differently than others. Such discrimination is significant only if the network does not have enough resources to

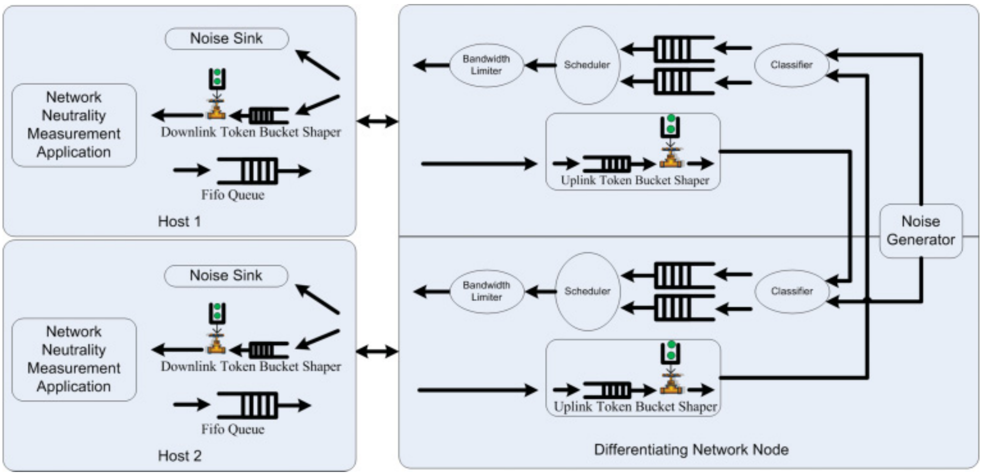
Noise Sink
Network Neutrality Measurement Application
Downlink Token Bucket Shaper
Fifo Queue
Host 1
Noise Sink
Network Neutrality Measurement Application
Downlink Token Bucket Shaper
Fifo Queue
Host 2
Bandwidth Limiter
Scheduler
Classifier
Uplink Token Bucket Shaper
Noise Generator
Bandwidth Limiter
Scheduler
Classifier
Uplink Token Bucket Shaper
Differentiating Network Node

Fig. 8. The testbed.

handle all of the traffic. Otherwise, the effects of the discrimination would be negligible. Therefore the discrimination has to be performed on a bottleneck link.

The path between two hosts on the Internet may have any number of bottleneck links; however, to make implementation easier, in this study, like in other studies dealing with network path emulation, only one bottleneck link was created [Sanaga et al. 2009]. Before traversing the link, packets are classified, queued in the egress queue appropriate to their class, dequeued by the egress scheduler, and transmitted on the link at the line rate. In the experiment design, all of these elements were recreated along with authentic cross traffic to add realistic noise to the measurement results.

The testbed was created using the Linux traffic control (tc) utility [Almesberger 1999] with Netem [Hemminger 2005] (see Figure 8). Note that in this design, a measurement packet sent by one of the hosts traverses the network elements in a manner identical to that of a real packet between two hosts on the Internet: measurement application → uplink token bucket → differentiating bottleneck node → downlink token bucket. Network elements that were not mentioned in this description operate in a FIFO fashion, with very little delay due to unconstrained bandwidth, and are therefore transparent.

The noise generator was created using a patched version of Tcpreplay [Turner], which replays traffic from a prerecorded traffic trace. To make the noise, seen in the differentiating node's queues, realistic, a real network traffic trace was used, specifically a 5 minute trace between 01:15 and 01:20 on April 15, 2010 from the Mawi archive. Using a prerecorded traffic trace to generate noise is not realistic, since noise does not react to the measurement traffic. However, when used in conjunction with a measurement technique that is not bandwidth intensive, the lack of reactivity does not affect the results. It simulates a realistic case of a network neutrality measurement between two hosts when the differentiation occurs on a high capacity backbone link, where the effect of the measurement traffic on the noise traffic is negligible.

5.2. Experiment Setup

The host ingress was configured with a downlink token bucket shaper with a rate of 15Mbps, a bucket size of 12Mbyte, and a maximum rate of 25Mbps. The host egress was a simple First In First Out (FIFO) queue. The differentiating node was configured with an ingress (uplink) token bucket with a rate of 1Mbps, burst size of 1Mbyte, and

Table I. Results of POPI Runs for Directions 1 (Top) and 2 (Bottom). Based on the ANR, POPI Groups the Traffic Classes into Service Tiers. The Grouping According to Tier is Shown in Parentheses Next to the ANR Calculated by POPI. For Example, in Short Experiment 4 POPI Classified All Three Classes to One Service Tier (Top), but Managed to Differentiate Them into Three Service Tiers in the Other Direction (Bottom)

Traffic Type	Short exp. 1	Long exp. 1	Short exp. 2	Long exp. 2	Short exp. 3	Long exp. 3	Short exp. 4
Port 22	0.870 (2)	0.802 (3)	0.760 (2)	0.817 (3)	0.771 (1)	0.829 (3)	0.786 (1)
Port 80	0.729 (2)	0.688 (2)	0.771 (2)	0.707 (2)	0.682 (1)	0.718 (2)	0.687 (1)
Port 8080	0.401 (1)	0.510 (1)	0.469 (1)	0.477 (1)	0.547 (1)	0.453 (1)	0.526 (1)
Port 22	0.865 (2)	0.913 (3)	0.859 (2)	0.919 (3)	0.875 (2)	0.912 (3)	0.943 (3)
Port 80	0.724 (2)	0.708 (2)	0.740 (2)	0.709 (2)	0.703 (2)	0.710 (2)	0.661 (2)
Port 8080	0.411 (1)	0.379 (1)	0.401 (1)	0.372 (1)	0.422 (1)	0.378 (1)	0.396 (1)

a peak rate of 2Mbps. After the shaping, frames are delayed for 10ms to emulate the delay caused by transmission and propagation. The egress is a simple strict priority scheduler with three classes, each with its own FIFO queue. The third (lowest) class is the default and the first two are "premium" classes. The parameters of the scheduler were set to provide three distinct levels of service to the packets replayed using Tcpreplay: ICMP and SSH packets (port 22), which are approximately 8.1% of all packets, receive the best treatment. HTTP packets (port 80), which are approximately 38% of all packets, receive the second best treatment. The rest of the traffic, 53.9% of all packets, receives the default treatment. The loss rates of the different classes are given in Table II. The parameters of the token buckets were chosen based on existing sizes and rates measured by ShaperProbe [Kanuparthy and Dovrolis 2011]. POPI was chosen to perform the neutrality measurements [Lu et al. 2010] since its source code is available on the project's website [POPI]. POPI was configured with the same parameters as in the experiments run by POPI's authors: using 32 bursts ($n_b = 32$) each of 40 packets ($n_r = 40$). It was configured to compare three classes of packets ($k = 3$), according to the three classes that were defined previously (TCP port 22 packets, TCP port 80 packets, and TCP port 8,080 packets, which are in the default class).

5.3. Experimental Results

POPI was executed through the differentiating node, without any noise in order to get a baseline measurement. It found no indication of differentiation in this case, since there was no packet loss in any traffic class. Afterwards, noise was introduced into the system using Tcpreplay and 14 more experiments (seven in each direction), were executed. Eight of the 14 experiments were executed with POPI's suggested settings, namely, $n_b = 32, n_r = 40$. The others were longer, using 150 bursts of 40 packets ($n_b = 150, n_r = 40$). The experiments were executed alternately with a short experiment followed by a long one, while the noise trace was replayed in a loop. The results of the experiments are shown in Table I.

Out of the eight short experiments (four in each direction), only one experiment detected all three classes correctly. Five more experiments concluded incorrectly that port 22 and 80 traffic belonged to the same class (2), while port 8080 was assigned to a lower class (1). Lastly, two short POPI experiments concluded that all three traffic types belong to the same class. Note that for the most part, POPI correctly gave higher ANR[2] scores to the high priority traffic; however, in many cases the difference in ANR was too small for POPI to correctly deduce the traffic classes. All six long experiments (three in each direction) were accurate, and correctly identified the three traffic types.

[2] ANR is a score POPI uses to order the traffic types according to the level of service it experiences [Lu et al. 2010].

Table II. Packet Loss Rates in the Differentiating Node: Without Manipulation and With Manipulation Using Two Different Threshold Values

Traffic	No	Threshold	
Type	manip.	10sec	60sec
Port 22	0.3%	0.28%	0.27%
Port 80	4.8%	3.78%	3.55%
Port 8080	8.17%	8.89%	8.96%

Next, we tried to skew POPI's results by using the observation from Section 4.1.2: the fact that a few pairs of IP addresses engage in more than a single session at a time. Since POPI's traffic appears as several concurrent sessions between a pair of IP addresses, it stands out and an ISP can choose to give it "special" treatment. To simulate what would happen in such a scenario, we run a monitoring program on the differentiating node that detected pairs of IPs that engaged in more than a single session at a time. It then added new classification rules to tc so all traffic between the detected pairs of IPs was placed in the lowest priority class.

The monitoring program processes packets on the fly and it does not know whether the current packet is the last of its flow. Therefore, if it does not see any packets of a flow for a threshold time, it assumes the flow is no longer active. We used two threshold values: an aggressive 10 second threshold, and a more relaxed 60 second value.

We failed to repeat the same POPI experiments since the differentiating node was not strong enough to match each outbound packet against the large amount of classification rules added due to the monitoring program. To overcome this, the process was modified using the following observation: from the rules that were added by the monitoring program, only one rule is added due to POPI's measurement; the rest of them redirect IP pairs that exist only in the noise trace. Since the noise trace is constant, it is possible to preprocess it offline and mark the packets that would have been redirected by the monitoring program. Note that this was just a cure to our low grade experimental setup, whereas in real networks, a faster network element is required to process all traffic. Once these packets are marked, one only has to add a single high priority rule to redirect all marked packets to the low priority class, to achieve the same effect as if tc could handle enough filters in real time. The monitoring program continues to work as before, but is monitoring only the pair of IPs that are actually running POPI.

Executing the same 14 POPI experiments again, this time with the monitoring and the preprocessed noise trace, gave a strong indicative result: not even one POPI experiment managed to detect any difference between the three traffic types. All 14 experiments concluded that all of the traffic was treated equally in the network although, in fact, only the measurement traffic was treated equally.

The obvious side effect of the manipulation was an increase in the amount of traffic handled by the lowest class due to false positives. With the 10 second threshold, 7.5% of packets changed classes due to the manipulation, all false positives. With the 60 second threshold, the number of false positives rose to 13.9%. This is not an insignificant rate of false positives, but it seems to be an acceptable rate considering the simplicity of the manipulation technique. This rate of false positives can be lowered using further classification with more targeted, possibly signature-based approaches that can scan the suspected 7.5%–13.9% of the packets.

The experiment results show that these false positives did not significantly alter the loss rates of the different classes. Apparently, this is because fewer high priority packets meant that the scheduler could service the low priority class longer. The loss rates of the different classes are shown in Table II. Note that both the rate of false positives and the packet loss depend heavily on the noise trace and the configuration of the differentiating node. Other noise traces and other configurations could produce different results. However, as this experiment shows, at least in some configurations, this trivial manipulation can invalidate the results of a network neutrality measurement.

6. COVERT MEASUREMENT DESIGN

Previous sections discussed and demonstrated the reasons that make current network neutrality measurement techniques detectable. Here, we present CONNEcT, the Covert Network Neutrality Measurement Technique, which is designed to deal with the defensive ISP model, as described in Section 3.2. The objectives of CONNEcT are as follows: (1) It must be difficult for an ISP to skew the measurements; (2) CONNEcT must be able to detect forged and modified packets; and (3) CONNEcT must be able to accurately measure a variety of performance metrics. As discussed in Section 4.1, active measurement techniques are fairly easy to detect, since the measurement traffic is different from other types of network traffic. On the other hand, passive techniques are limited in what they can measure and how accurate those measurements are. This seems to create a trade-off between detectability and accuracy, which CONNEcT is designed to exploit.

6.1. Overview

Hosts running CONNEcT that wish to initiate a measurement slightly modify packets that are sent as part of the normal host operation, and look for a response. Namely, the hosts start a covert communication using a covert handshake, which we are the first to describe (see Section 6.3.1). Once hosts identify each other, they use the existing flow of packets between them to secretly pass information regarding the traffic being sent. The majority of information sent over the covert channel consists of samples; each sample corresponds to a previously sent packet.

Samples are four bytes long and contain a control field (two bits), a timestamp (12 bits), a sequence number (16 bits), and an authentication signature of a packet (only two bits). They are used by the receiving host to check packets for authenticity as well as measure several network metrics, as described in Section 6.4. A small portion of the covert information contains control words that are used for parameter negotiation and session management.

Periodically, CONNEcT reports its findings to a main server for aggregate statistical analysis. Note that unlike Switzerland, in CONNEcT hosts share measurement information directly with each other. This means that reports to the main server are relatively small, aiding in their concealment.

6.2. Background on Covert Channels

Covert channels are a means of hiding messages between two parties in plain sight. Simmons [1983] modeled them with the following prisoner problem: two prison inmates, Alice and Bob, are interested in exchanging information with each other, but are forced to communicate via written messages that are passed through the prison warden, who can read and even alter the messages to reveal hidden information. Alice and Bob are forced to find a means of hiding covert information inside the plain messages being passed. These methods for passing hidden information in "normal" messages are often referred to as a *covert channel*. The network security literature covers a multitude of covert channels [Zander et al. 2007].

There is a clear trade-off between the two main characteristics of a covert channel: its covertness, namely, how hard it is to detect it, and its capacity. CONNEcT can operate with any well hidden covert channel: while medium to high capacity channels that can carry, on average, at least several bytes of covert information per packet are preferred, lower capacity channels can also be used at the expense of accuracy by reducing the number of samples passed between the hosts.

6.3. CONNEcT's Covert Channel

CONNEcT's covert channel assumes that there is a shared secret S between the participating hosts. The means by which the hosts establish a shared secret will be discussed in Section 6.3.1. Assuming that the covert information to be sent is B, using this shared secret S, the hosts encrypt the covert information to be transferred: $C = E_S(B)$, with E_S is an encryption function with the key S and C is the resulting cipher text. This is done mainly for obfuscation purposes, so that the sent covert information will not have a fixed recognizable format. Then, using the shared secret S, the sender chooses where to embed the covert information in the sent packet. The embedding always takes place in the application layer payload, since the headers contain well-known values whose alteration might cause suspicion. An offset from the payload part of the transport layer is calculated, based on S and the value of the IP identification field, and the covert data is placed at that offset in the packet. The offset, I, is thus calculated as follows:

$$I = \left\lfloor \frac{H_S(IPID)[0:15]}{2^{16}} \cdot (P_{len} - I_{min} - C_{len,max}) \right\rfloor + I_{min},$$

where P_{len} is the length of the payload after the insertion of the covert information; $C_{len,max}$ is the maximum length of the covert information that can be embedded in a given packet; and the term I_{min} is the minimal offset used to avoid embedding in the application layer headers. The term $\frac{H_S(IPID)[0:15])}{2^{16}}$ designates bits 0 through 15 of the result of a cryptographic hash function H on the IP identification value of the packet IPID with key S divided by 2^{16}. Assuming the values of the hash function bits are uniformly distributed, this term outputs a uniformly distributed value in the interval [0,1), so that both parties running CONNEcT can calculate. The value of $C_{len,max}$ depends on P_{len} as described in the following, so longer packets can hold more covert information. The exact value is calculated as $C_{len,max} = \lfloor P_{len} \cdot Embedding\% \rfloor$, and $C_{len,max} > P_{len} - I_{min}$ where $Embedding\%$ is the portion of covert information that is allowed to carry covert data. $Embedding\%$ should be set to a fairly low value to maintain covertness. For example, setting $Embedding\% = 1\%$ and using a packet with a 1,000 byte payload (P_{len}) with no application layer headers ($I_{min} = 0$) can embed up to 10 bytes of covert information, resulting in a 1,010 byte packet. The index of the embedding is determined by the hash function and is guaranteed to be between 0 and 999.

The content of the embedded information is divided into groups of various sizes. The first two bits of each group serve as control bits: the first indicates whether the payload of the group is a sample or a control word; the second bit indicates whether the current group is the last group of the packet. Control words have a variable size and are used for parameter negotiation and message passing between the parties that run CONNEcT. The structure of the control messages will not be elaborated on. Samples, which are the bulk of the covert information, are four-byte-long groups. Each sample carries information about a previously sent packet. They are used by the receiver to check for the authenticity of the received packets, and perform measurements of several network metrics as discussed in Section 6.4.

Figure 9 shows how the samples are embedded in the message, and the sample structure. The 16 bit sequence number field is large enough for the sender to be able to distinguish up to 65,535 previously sent packets. The timestamp, encoded in millisecond resolution, is enough to encode time differences of up to 4,095ms. The two-bit authentication signature is composed of the last two bits of the cryptographic

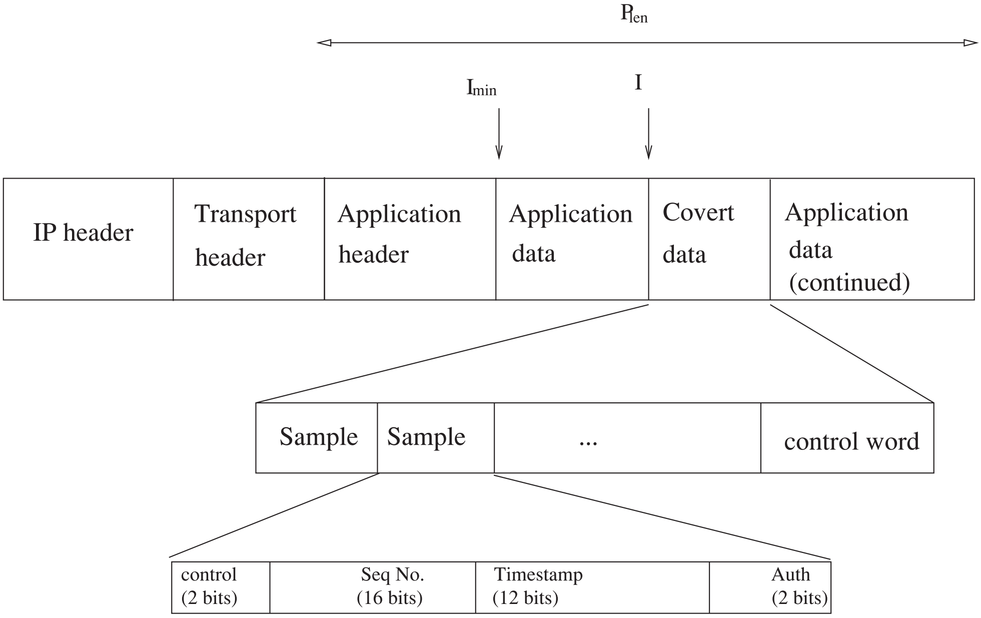

Fig. 9. CONNEcT's covert channel embedding (not to scale).

hash with key S on the payload of the packet. The way CONNEcT uses these fields is detailed in Section 6.4.

To save covert information, CONNEcT does not require a sample for every sent packet. Instead, it can trade accuracy for covertness by reducing the number of sent samples. For this, hosts running CONNEcT can use control words to negotiate a sampling threshold $SampThresh$ between 0 and 1. The hosts will then send samples only of packets whose $\frac{H_S(IPID)[0:15]}{2^{16}} < SampThresh$.

6.3.1. Covert Handshake. The covert handshake is used by hosts running CONNEcT to secretly identify each other and establish a shared secret, S, which is needed for the creation of a common covert channel. The covert handshake is based on the idea of "proof of work," introduced by Dwork and Naor [1992] in the context of fighting spam. This proof of work uses a pricing function that is hard to calculate but easy to verify. In the covert handshake, both hosts perform a proof of work at the beginning of a session between them. This cost is minor and can be performed easily. However, performing the calculations in bulk in order to identify the handshakes in every session between any pair of hosts in a network is expensive.

CONNEcT's proposed pricing function is as follows: given a predefined cryptographic hash function H and a set of values D, which both hosts agree on during the start of their session (e.g., IP addresses and port numbers), both clients find an integer value X such that

$$\arg\min_X H(D+X) \leq HTresh,$$

where $HThresh$ is a predetermined threshold value. The only way to find the value X is to try all possible values starting with 1. Once found, X becomes the common shared secret S, as both hosts will find the same value. The difficulty of the pricing function is determined by $HThresh$, which is tuned so that X could be calculated within several milliseconds on modern computers.

Consider the TCP handshake between a client and a server. When a server receives a SYN, it knows the entire set of values D, based on the SYN packet and knowledge of its own response. It can therefore calculate S. When the SYN-ACK is received by the client, D is known to it as well. The server now secretly informs the client that it is running CONNEcT. It does so by encrypting the IP identification field value from the SYN packet using S, and placing this value in the IP identification field of the first data packet it sends to the client. In response, the client, knowing that the server is running CONNEcT, sends the server a predetermined preamble over the covert channel, established using S. This informs the server that the client is running CONNEcT as well.

This use of the IP identification field imitates the default behavior of this field in Linux. In Linux, packets in a TCP flow start off with a random IP identification value in each direction. After the initial value is set, it is incremented by 1 with every sent packet. The client's initial, random, identification number appears in the SYN packet sent to the server. The server's initial IP identification appears in the first data packet, after the three-way handshake. The SYN-ACK sent by the server has an identification value of zero. The behavior of this field in UDP sessions is slightly different; the first packet in both directions contains a random identification value, and no packet has an identification of zero. As a consequence, in UDP, the covert handshake is shorter by one packet; the SYN-ACK is skipped.

In this scheme, if one of the hosts was not running CONNEcT, it would not have noticed any abnormal behavior from the host running CONNEcT. Moreover, the host running CONNEcT would have known that the second party is not running CONNEcT.

6.4. Measuring Neutrality

6.4.1. Packet Authenticity. Unlike all existing network neutrality measurement techniques, except Switzerland, CONNEcT checks for the authenticity of sent and received traffic. For each sent packet, a short authentication signature is calculated by the sender and placed in a sample. When the sample is received by the second party, it compares the signature from the sample with the signature it calculates from the received packet. Due to the signature's short length, given a uniform distribution of bits in the signature, a portion (a quarter in our two bit design) of all modified packets may wrongly pass this test, but any systematic modification of packets will be eventually detected.

6.4.2. Packet Loss and Injected Packets. CONNEcT uses the value of the IP identification field to match packets to samples and to calculate packet loss. This is done by placing the IP identification value into the "sequence number" field of the sample describing that packet. A receiving host can then use these sequence numbers to detect lost packets and spurious packets, injected by the network.

6.4.3. Delay. CONNEcT measures the network delay between two hosts running it using the 12 bit timestamp field included in each sample. This field contains the low 12 bits of the time in milliseconds when the packet that corresponds to the sample was sent. By comparing it to the receiver's local time, delay can be measured as discussed in this section.

CONNEcT supports two different approaches of delay measurement. Given that hosts are aware of their own capabilities and are able to covertly exchange information with each other, they can negotiate the best measurement strategy. Between hosts with precise time synchronization, such as phones with GPS receivers, CONNEcT can measure the one-way delay of packets in each direction. This is done by subtracting the timestamp encoded in a sample from the time when the corresponding packet was observed, yielding accurate measurements of one-way delays that are under 4,096ms (using a 12 bit field).

Hosts that do not have a GPS receiver or a similar technology rely on NTP-like measurements to make accurate assessments of the RTT. These assessments require hosts to know how much time has passed between packet transmissions by the second host. To keep track of these times, hosts use the 12 bit timestamps provided by the other party. To be able to track transmission times for periods longer than 4,096ms, hosts have to keep track of the 4,096ms periods of the other party. For this, hosts keep a virtual clock that follows the other host's clock. The virtual clock is initialized by the first packet a host receives that has a corresponding timestamp.

Assume host A sends packets to host B. The first packet that has a timestamp is denoted as packet 1. Using packet 1, host B initializes its virtual clock to match host A's clock. Due to the propagation delay of packet 1, B's virtual clock is initialized Δtms after A's clock. Denoting the propagation time of the ith packet by d_i, then the difference between the clocks is $d_1 = \Delta t$.

Denote by t_i and r_i the times packet i was sent and received, respectively, each measured by the local clock. Therefore, $r_i = t_i + d_i - \Delta t = t_i + d_i - d_1$. Note that by definition $d_i \geq 0\, \forall i$. The timestamp of the ith packet, T_i, can be written as $T_i \equiv t_i \pmod{4{,}096}$. Therefore, there is a natural k for which $t_i \equiv T_i + 4{,}096k$ holds. Host B is inferring t_i from the timestamp T_i and r_i, by looking for a virtual time $vt = T_i \pmod{4{,}096}$ that is the closest to the observed r_i. Formally, $t_i = \arg\min_{vt} |r_i - vt|$ such that, $vt = T_i \pmod{4{,}096}$. This gives the correct t_i as long as $|r_i - t_i| < 2{,}048$ms, namely, when $|d_i - d_1| < 2{,}048$ms. In other words, the estimation of the RTT will be accurate as long as all one-way delays across the network are shorter than 2,048ms, which is almost always the case in the current Internet.

Finally, to calculate RTT, hosts use control words to periodically report to each other the local time at which they received a specific packet. This provides the second hosts with enough information to calculate the round-trip time using standard techniques. Note that in order to prevent clock drift during long sessions, this initialization is performed periodically to synchronize the virtual clock to the actual clock.

6.4.4. Path Capacity. Unlike Glasnost and NANO, CONNEcT does not measure the throughput of observed flows. Instead, it measures path capacity, defined as the IP layer transmission rate of the narrowest link along the path. Path capacity is one of the main factors that affects TCP throughput, and since traffic shapers change the capacities of paths, measuring path capacities could directly detect shapers that limit the throughput of certain applications or users.

CONNEcT relies on a passive capacity estimation technique called PPrate [En-Najjary and Urvoy-Keller 2006], created to measure path capacities using packet traces. It is based on the packet pair technique, or more precisely on an active capacity measurement technique called Pathrate [Dovrolis et al. 2004] that uses the packet pair technique.

To perform a capacity measurement, PPrate searches the packet trace for packet pairs that were sent back to back. From each packet pair, it calculates the estimated capacity of the path; the distribution of these estimates is then analyzed. Generally, this distribution is multimodal [Dovrolis et al. 2004] and one of the modes represents the true capacity of the path. To choose the correct mode, PPrate groups packet pairs into packet trains of N packets. From these trains it again estimates the path capacity and creates a second distribution. PPrate increases N until this distribution is unimodal. This remaining mode is the *Asymptotic Dispersion Rate* (ADR). Finally, PPrate estimates the path capacity as the strongest and narrowest mode among those larger than the ADR. In a comparison between several passive capacity estimation tools, PPrate was found to be the most accurate in its estimations in both laboratory experiments and when used on actual Internet paths, while requiring as few as 300 sampled packets to achieve results close to those of active capacity estimation tools [En-Najjary and Urvoy-Keller 2008].

PPrate does have some drawbacks stemming from its passive approach. As PPrate uses one-sided packet traces for capacity estimation, it is forced to guess which packets were sent back to back, and therefore can be used as samples; mistakes in these guesses could lead to large inaccuracies. CONNEcT avoids the guesswork by passing packet timestamps between the two sides.

7. SUMMARY AND FUTURE WORK

We showed by analysis and experimentation that network neutrality measurements, as suggested by all previous active measurement works, including POPI, Glasnost, NetPolice, and DiffProbe, can be easily manipulated. We then suggested CONNEcT, a new network neutrality measurement that uses the idea of covert channels and the newly proposed idea of the covert handshake to secretly perform network measurements without being detected by the traffic manipulator.

A future research direction is to examine the means by which CONNEcT reports its measurements to a main server. Since CONNEcT passes most measurement information directly between hosts, very little data has to be reported, making the reports fairly easy to hide, for example, using techniques mentioned in Section 4.2.

REFERENCES

W. Almesberger. 1999. Linux network traffic control—Implementation overview. In *Annual Linux Expo*.

R. Beverly, A. Berger, Y. Hyun, and k. Claffy. 2009. Understanding the efficacy of deployed internet source address validation filtering. In *Internet Measurement Conference*. ACM, 356–369.

CAIDA. 2011. Anonymized Internet Traces 2011 Dataset. Retrieved from http://www.caida.org/data/passive/passive_2011_dataset.xml.

I. Cunha, R. Teixeira, D. Veitch, and C. Diot. 2014. DTRACK: A system to predict and track internet path changes. *IEEE/ACM Transactions on Networking* 22, 4 (2014), 1025–1038.

Default Time To Live (TTL) values. 2009. Retrieved from http://www.binbert.com/blog/2009/12/default-time-to-live-ttl-values/.

M. Dischinger, M. Marcon, S. Guha, K. P. Gummadi, R. Mahajan, and S. Saroiu. 2010. Glasnost: Enabling end users to detect traffic differentiation. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation (NSDI)*. 27–27.

M. Dischinger, A. Mislove, A. Haeberlen, and K. P. Gummadi. 2008. Detecting BitTorrent blocking. In *Internet Measurement Conference (IMC)*. USENIX/ACM, 3–8.

C. Dovrolis, P. Ramanathan, and D. Moore. 2004. Packet-dispersion techniques and a capacity-estimation methodology. *IEEE/ACM Transactions on Networking* 12, 6 (2004), 963–977.

C. Dwork and M. Naor. 1992. Pricing via processing or combatting junk mail. In *Proceedings of CRYPTO'92*. 139–147.

Electronic Frontier Foundation (EFF). 2010. Switzerland Network Testing Tool. Retrieved from https://www.eff.org/ pages/switzerland-network-testing-tool.

T. En-Najjary and G. Urvoy-Keller. 2006. PPrate: A passive capacity estimation tool. In *Proceedings of the 4th IEEE/IFIP Workshop on End-to-End Monitoring Techniques and Services*. 82–89.

T. En-Najjary and G. Urvoy-Keller. 2008. Passive capacity estimation: Comparison of existing tools. In *Proceedings of the Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'08)*. IEEE, 346–352.

P. Gill, M. Crete-Nishihata, J. Dalek, S. Goldberg, A. Senft, and G. Wiseman. 2015. Characterizing web censorship worldwide: Another look at the OpenNet initiative data. *ACM Transactions on the Web* 9, 1 (Jan. 2015).

WAND Group. 2010. Waikato I trace. Retrieved from http://www.wand.net.nz/wits/waikato/1/waikato_i.php.

S. Hemminger. 2005. Network emulation with NetEm. In *Proceedings of the 6th Australia's National Linux Conference*. 18–23.

International Organization for Standardization. 2006. *ISO 3534-2:2006 Statistics—Vocabulary and Symbols—Part 2: Applied Statistics*.

P. Kanuparthy and C. Dovrolis. 2010. Diffprobe: Detecting ISP service discrimination. In *Proceedings of IEEE INFOCOM*. 1–9.

P. Kanuparthy and C. Dovrolis. 2011. ShaperProbe: End-to-end detection of ISP traffic shaping using active methods. In *Proceedings of the ACM IMC*. 473–482.

G. Lu, Y. Chen, S. Birrer, F. E. Bustamante, C. Y. Cheung, and X. Li. 2007. End-to-end inference of router packet forwarding priority. In *Proceedings of IEEE INFOCOM 2007*. 1784–1792.

G. Lu, Y. Chen, S. Birrer, F. E. Bustamante, and X. Li. 2010. POPI: A user-level tool for inferring router packet forwarding priority. *IEEE/ACM Transactions on Networking* 18, 1 (2010), 1–14.

MAWI working group. 2010. Anonymized packet traces without payload: WIDE-TRANSIT link. Retrieved from http://mawi.wide.ad.jp/mawi/ditl/ditl2010/.

POPI Website. Homepage. Retrieved from http://list.cs.northwestern.edu/popi/.

SAN. 2008. Sandvine's Reply Comments in connection with Telecom Public Notice CRTC 2008-19, Review of Internet Traffic Management Practices of Internet Service Providers. Retrieved from http://crtc.gc.ca/public/partvii/2008/8646/c12_200815400/1109151.pdf.

P. Sanaga, J. Duerig, R. Ricci, and J. Lepreau. 2009. Modeling and emulation of internet paths. *NSDI* 9 (2009), 199–212.

A. Sfakianakis, E. Athanasopoulos, and S. Ioannidis. 2011. CensMon: A web censorship monitor. In *Proceedings of the USENIX Workshop on Free and Open Communications on the Internet*.

G. J. Simmons. 1983. The prisoners' problem and the subliminal channel. In *Proceedings of Crypto'83*. 51–67.

M. B. Tariq, M. Motiwala, N. Feamster, and M. Ammar. 2009. Detecting network neutrality violations with causal inference. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*. ACM, 289–300.

Aaron Turner. Tcpreplay. Retrieved from http://tcpreplay.synfin.net/.

T. Wu. 2003. Network neutrality, broadband discrimination. *Journal of Telecommunications and High Technology Law* 2 (2003), 141.

S. Zander, G. Armitage, and P. Branch. 2007. A survey of covert channels and countermeasures in computer network protocols. *IEEE Communications Surveys & Tutorials* 9, 3 (2007), 44–57.

Ruishan Zhang, Xinyuan Wang, Ryan Farley, Xiaohui Yang, and Xuxian Jiang. 2009b. On the feasibility of launching the man-in-the-middle attacks on VoIP from remote attackers. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security (ASIACCS)*. 61–69.

Y. Zhang, Z. M. Mao, and M. Zhang. 2009a. Detecting traffic differentiation in backbone ISPs with NetPolice. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*. ACM, 103–115.

Z. Zhang, Y. Zhang, Y. Charlie Hu, Z. Morley Mao, and R. Bush. 2010. iSPY: Detecting IP prefix hijacking on my own. *IEEE/ACM Transactions on Networking* 18, 6 (Dec. 2010), 1815–1828.

Received December 2015; revised August 2016; accepted November 2016