Approximating a Global Passive Adversary Against Tor

Sambuddho Chakravarty
Columbia University, NY
sc2516@cs.columbia.edu

Angelos Stavrou
George Mason University, VA
astavrou@gmu.edu

Angelos D. Keromytis
Columbia University, NY
angelos@cs.columbia.edu

Abstract

We present a novel, practical, and effective mechanism for exposing the IP address of Tor relays, clients and hidden services. We approximate an almost-global passive adversary (GPA) capable of eavesdropping anywhere in the network by using LinkWidth. LinkWidth allows network edge-attached entities to estimate the available bandwidth in an arbitrary Internet link without a cooperating peer host, router, or ISP. By modulating the bandwidth of an anonymous connection (e.g., when the destination server or anonymous client is under our control), we can observe these fluctuations as they propagate through the Tor network and the Internet to the end-user's IP address. Our technique exploits one of the design criteria for Tor (trading off GPA-resistance for improved latency/bandwidth over MIXes) by allowing well-provisioned (in terms of bandwidth) adversaries to effectively become GPAs.

Although timing-based attacks have been demonstrated against non-timing-preserving anonymity networks, they have depended either on a global passive adversary or on the compromise of a substantial number of Tor nodes. Our technique does not require compromise of **any** *Tor nodes or collaboration of the end-server (for some scenarios). We demonstrate the effectiveness of our approach in tracking the IP address of Tor users in a series of experiments. Even for an under-provisioned adversary with only few network vantage points, we can identify the end user (IP address)/hidden servers in many cases.*

1 Introduction

Network anonymity schemes such as Tor [6] and Tarzan [9] represent an important point in the design space, trading off resistance to specific attacks for improved performance, improving their acceptability by users. By design, such systems are vulnerable to a Global Passive Adversary (GPA) that can correlate traffic flows seen in different links [23] by foregoing the computation and bandwidth-heavy traffic padding schemes typically used to thwart traffic analysis [12, 13, 30]. This is considered an acceptable tradeoff for many usage scenarios: these schemes offer adequate protection against all but a determined (and possibly targeted) attack by a GPA, of which there are very few, and who could in practice use other surveillance methods.

In practice, there exist a number of attacks against these systems that typically leverage a small number of compromised network entities to fully or partially expose information about a user of these systems [1, 3, 14, 19, 31]. Despite these attacks, it is generally believed that consistent, fine-resolution tracking of end users[1] of these anonymity systems is impractical for the large majority of users and organizations. We demonstrate that this belief is unfounded, by presenting a novel and effective approach for performing user trace-back through an anonymous circuit. For the remainder of this paper we will focus on Tor, both for concreteness and because of its large deployment and user base. However, our approach is equally applicable to other similar systems that do provide strong protection against traffic analysis.

Briefly, our approach uses *LinkWidth*, a novel *single-end* available-bandwidth estimation tool to identify the Tor nodes involved in a Tor circuit of interest, and to then trace-back to the IP address of the end user or Hidden Server. LinkWidth allows us to estimate the available bandwidth on an arbitrary network link *without* direct access either to that link itself or to an appropriately positioned cooperating host. The Tor nodes and the links between these and the client host or Hidden Services are identified by using LinkWidth to detect induced traffic fluctuations on the anonymous connection to a server of interest. These fluctuations

[1]We will use the terms "user" and "end user" to refer to the IP address of the host acting as a Tor client. Although in many cases this can lead to the identification of a specific user/owner of the machine, we recognize that network artifacts such as NAT can obscure such associations.

can be created by the server itself (if it is cooperating with the attacker), by a router/node close to the server (*e.g.,* when collaborating with the server's ISP or otherwise hijacking/compromising/legally compelling use of such a node), or by launching a targeted network denial of service attack against the appropriate link(s), router(s) or the server itself.

In our experiments, we assumed that the attacker controls either the server or the client; although any other scheme for causing "large enough" traffic variations, as defined later in the paper, would suffice. Our scheme enables an attacker with access only to a few high-bandwidth edge nodes (hosts) and a map of the network to effectively act as a global passive adversary in the Tor threat model. A larger number of distributed measurement nodes ("vantage points") enables for more complete coverage of the network links, while high-bandwidth connectivity is needed for measurement accuracy, as we describe in Section 4. We stress that we do not assume that the attacker has access to large numbers of routers, network infrastructure nodes (*e.g.,* DNS or DHCP servers), or Tor nodes, nor do we exploit software vulnerabilities that inadvertently expose the true network identity of the user.

We built a prototype of LinkWidth and evaluated its effectiveness in detecting small variations in available bandwidth in a series of experiments in a lab environment. We then used our prototype to launch a traceback attack against a number of connections, through Tor, to a client or server under our control. In our experiments, we only had access to few network vantage points, representing an under-provisioned adversary. Even in that case, we could accurately identify the end user in many cases. With proper process coordination, a well-provisioned adversary can hope to complete an attack in less than 20–30 minutes. Each step of the process (sensing one link) took 25–30 seconds on average, with a worst case of 50 seconds. Possible countermeasures include shorter circuit lifetimes, limited traffic smoothing by Tor nodes, use of multiple parallel circuits to access the same server, and preventing the use of long-lived connections. We note that the use of longer Tor circuits does not appear to make the attack more difficult.

The novel contributions of this paper include:

• A practical and effective attack against Tor and similar anonymity systems. A bandwidth-provisioned adversary can trace-back through a Tor circuit and expose the network identity of a Tor user or Location Hidden Service. Using a map of the network-path, a well-provisioned adversary, can determine the subnetwork to which a client or Location Hidden Service belongs to.

• *LinkWidth*, a novel single-end available-bandwidth estimation technique.

• An implementation of LinkWidth and of the mechanism for performing traceback through the existing Tor network.

• Experimental demonstration of the feasibility and effectiveness of our attack, and a first characterization of the important parameters of our scheme and its limitations.

• A discussion of possible practical countermeasures.

Report Organization We continue with an overview of related work in attacks against Tor-like anonymity systems and on bandwidth estimation techniques, highlighting the differences with our work. Section 4 describes LinkWidth, and its use in performing trace-back through Tor and the Internet. We analyze the accuracy of LinkWidth in Section 5, and provide experimental evidence of the feasibility and effectiveness of our attack in Sub-section 5.1. We discuss possible countermeasures in Section 6, and conclude the paper with our plans for future work in Section 7.

2 Related Work

Onion-routing anonymizing networks [28] use multihop encrypted communications to protect sender and/or receiver anonymity. Tor extends the existing onion routing scheme by adding support for integrity protection, congestion control, and location-hidden services through rendezvous points. An adversary observing all links in an onion routing network can record arrival and departure times for all messages in the network and use statistical methods to determine exactly who is communicating with whom [22, 25, 31]. However, Tor is considered "good enough" in practice for semi-interactive traffic, *eg.* web sessions, because few entities are believed to have the ability to act as global passive adversaries. This is precisely the assumption that our work attacks.

Our approach uses single-ended bandwidth and throughput estimation to expose anonymity of Tor clients and servers. Though novel, there have been prior efforts in using network latency to attack Tor . Murdoch *et al.* [19] focus on using network latency to determine if a relay node is a part of a specific Tor circuit. Their method requires a server to send pseudo-random data as fast as allowed by the underlying network to the victim client. The adversary uses a modified Tor Proxy for establishing single-hop circuits (rather than the default 3 hops) through the victim Tor relay, back to itself. The corrupt server sends traffic to the client having a particular "on-off" pattern. The adversary attempts to observe the variation in one-way delay through the victim Tor relay due this induced network traffic fluctuation. Higher correlation between these induced fluctuations and the observed one-way latency distortions gives a better probability that the victim Tor relay is the one which is a part of the victims Tor circuit. *This technique is however limited to only uncovering Tor relays participating in a Tor circuit.*

Hopper *et al.* [14] go a step ahead and try to use a combination of this technique and pairwise round-trip times (RTTs) between Internet nodes as input to statistical measures to correlate Tor nodes to probable clients. In addition, their method can be extended to application-layer RTT estimates (rather than TCP RTT estimates). *However we argue that RTT is a temporal network parameter which cannot be used as constants.*

Burch and Cheswick [4] proposed the use of targeted denial of service attacks for trace-back of a DoS source that used IP spoofing. Their approach was to cause interference with remote routers such that, when targeting the correct router/link, they would notice fluctuations in the attack-DoS traffic. Using a network map and an iterative trace-back process similar to ours, they would eventually identify the source of a DoS, or at least its approximate location (*e.g.,* hosting ISP).

All such attacks require observable traffic which is either measured offline or generated by Tor insiders. But, there are hidden assumptions about the network path conditions between the adversary and the relay(s). These include one-way packet delays, round-trip time, bottleneck capacity, throughput and packet jitter, which are assumed to be either constant [14] and/or accurately manipulated by Tor insiders [19] that participate in the victim's circuit. These assumptions are very restricting and can lead to a large number of false positives and false negatives.

In contrast, our approach is much less invasive and doesn't require the inclusion of a malicious Tor relays, padding, extra traffic, and nominal network conditions (*i.e.,* no congestion) and can be used to identify not only Tor relays involved in a circuit but also specific client (in terms of their IP address). Our technique relies on measurement of available bandwidth and/or throughput between an outsider (the GPA) to the the IP routers and Tor relays involved in the Tor circuit. For doing this we implemented a tool which we call *LinkWidth.* LinkWidth, which emulates a TCP Westwood sender to measure the available bandwidth and/or throughput, requires no support from a peer. The sender alone can be used to measure the available bandwidth and/or throughput of the path connecting itself to the server.

We use remotely located network vantage point(s) to measure the fluctuations of available bandwidth/throughput to the Tor relays and link these fluctuations to client's/server's communication. Our only assumption is that the vantage network point is a host that has higher bandwidth connection compared to the path that leads the victim client to the victim relay/router with path bottleneck lying as close as possible to the victim relay/router.

Our approach is that of a "real-time" eavesdropper, equipped with a map of the Internet. He/She "traces" bandwidth fluctuation on routers connecting a Tor relay (Entry Node) to a client or hidden service. These fluctuations are introduced by a corrupt client/server.

There may be situations where it may not be possible to probe for bandwidth fluctuation on routers; mostly due to lack of acknowledgment reply packets. An adversary, equipped with a map, may still learn about possible subnetworks or Autonomous Systems (AS) from which the anonymous traffic is originating. Assuming the IP route between the anonymous client/server and his/her Entry Node stays fixed for most "large" duration, our technique can be used to determine subnetwork or AS to which a client/server belongs to with high probability.

Previous work also disregarded traffic filtering and shaping either at the end hosts or at network edges. Our scheme uses TCP packets to probe the link capacity/throughput. Where TCP is filtered or rate-limited, we emulate the same behavior using ICMP. Unlike previous work, we assume least control over various network elements. All such traffic analysis attacks may fail if the Tor relays perform traffic engineering by controlling the outgoing traffic rate and burst length. This is discussed at the end of this paper.

2.1 Bandwidth Estimation

Prior research in bandwidth measurement has taken two major forms [24]. One focuses purely on the measurement of bottleneck bandwidth for IP payload. Examples include Pathchar [16], Pathrate [7] and Pchar [8]. These techniques rely on the *Packet Pair Technique* [17]. A pair of packets, sent back-to-back to the destination, "spreads" in time. This spreading in time, known as *received dispersion*, is inversely proportional to the bottleneck link capacity. The capacity is thus measured as $B = L/T$. In this formula, L is length of the second transmitted packet (in bits). T, the dispersion, is measured as the latency between the reception of the last bit of the first packet and the last bit of the second packet. The *Packet Train Technique* extends the Packet Pair Technique by sending a train of packets. The use of more packets minimizes the error due to noise and cross traffic. A detailed discussion of various packet pair and packet train techniques and their comparison can be found elsewhere [21].

The other major family of measuring techniques focuses on the estimation of end-to-end throughput, typically for use with transport-layer protocols such as TCP. The transport-protocol mechanics are geared towards optimizing in-order and correct delivery of messages in the presence of unreliable links without underutilizing the end-to-end path capacity. Therefore, it is important for TCP to determine the number of bits correctly received since the previously received acknowledgment. Tools such as Iperf [29] and abget [2] and Sprobe [26] come close to measuring throughput.

TCP Westwood [10] and its variant, TCP Westwood+ [11] use average throughput to correctly estimate how to modify the *slow start threshold*.

3 Overview of Tor Architecture

Tor [6] is a popular and widely used client/server anonymity system. It is geared towards providing the users high performance for *semi*-interactive Internet application. Tor can be used for both *initiator* and *responder* anonymity. Initiator anonymity is hiding the true identity (IP address) of the client. Responder anonymity allows a server to provide a TCP service without revealing its IP address. In this section, we provide overview of the architecture and design of Tor only from the perspective that is relevant to highlight our work.

From a computer networking point of view, Tor can be viewed as an overlay network of application layer proxies or Onion Routers (ORs). The Tor model also includes Rendezvous Points (RP) and Introduction Points, essentially for supporting responder anonymity. [2]

Tor Circuits : Tor circuits are formed using three ORs (by default). The first hop is known as the *Entry Node* in Tor terminology, the second is the *Middleman* and the third is the *Exit Node*. An OP uses the public-keys (Onion Keys) of the three ORs to to establish a shared secrets with them. The OP thereby encrypts the payload (512 byte units called *cells*) first using the shared secret of the final Exit Node, followed by that of the Middle Man and finally by those of the Entry Node's. This technique, common for many anonymizing networks and mixes, is known as *Telescopic Encryption*. Each of the three ORs, in-effect "peels-off" the headers off the Tor cell and forward it to the next OR along the circuit. The Exit Node decapsulates and decrypts the Tor cell and obtains the payload; which is sent encapsulated into a regular TCP/IP header to the appropriate peer. Since an OP picks up different ORs for every new circuit, the peer host sees a different source IP address each time a new circuit is established.

[2]These anonymized responders are known as *Location Hidden Services.*

Figure 3 shows how a circuit with three ORs is negotiated. The Tor OP keeps a time window within which all connections are given to the same circuit. This is used to prevent the adversary from linking new requests to earlier actions.

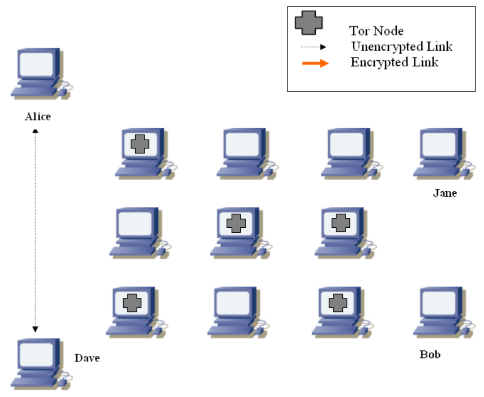

Figure 1. Alice's Tor client obtains a list of Tor nodes directly from a directory server

The original Onion Routing builds a separate circuit for each TCP stream. This fact made it easier for the adversary to link TCP streams to separate circuits. To counter this, Tor enforces sharing of the same Tor circuit by many TCP connections.

Hidden Services : Tor supports responder anonymity through Hidden Services. Responder anonymity allows a server to provide a TCP service without revealing its IP address. Hidden Services prevent against attacks that require IP address of the server. In this section we present an overview of how Hidden Service work.

Generally, service URI to IP address translation is done using the Domain Name System (DNS). For a Hidden Service, the regular DNS name used within the TCP/IP model, is replaced by a pseudo-random string (derived from the long-term public key of the server) ending with ".onion" domain name. A query to resolve a service URI ending in ".onion", can be resolved **only within the Tor network**. This new URI and the long-term public-key, representing the service, is published by the server, the first time it joins the Tor network. Only an insider can thereby access the service through an anonymous Tor circuit connecting himself to Hidden Service. This is shown in figure 3.

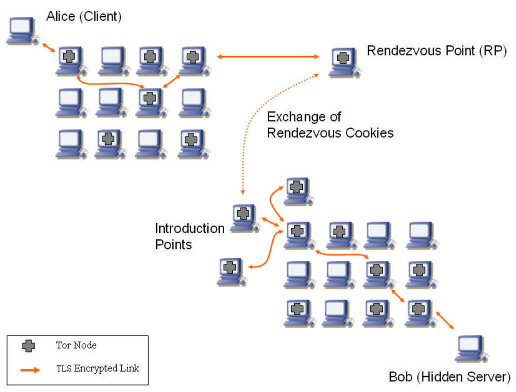

Figure 2. Alice's Tor client negotiates a connection with the Hidden Sever (Bob) across her Rendezvous Point(RP) and Bob's Introduction Points

A form of Diffie-Hellman key exchange ensues between the client and the Hidden Server through ORs known as *Rendezvous Point (RP)* and *Introduction Points*. The outcome of this exchange is the joining of the client and the server circuits, and hence the establishment of an anonymous communication channel between the client and the server. A detailed description of this handshake procedure is presented in the Tor design document [6]

Threat Model : A detailed description and analysis of the threat model considered by Tor and the possible mitigation strategies is beyond the scope of this paper. We rather focus on presenting a mechanism that attempts to approximate the effect of global adversary by observing the network traffic and correlating the victim's circuit to the ORs that it likely uses [3]. The original Tor model does aim to mitigate such *traffic analysis* attacks. We only try to approximate a global passive adversary (which may be anyways a difficult task due to limited control over network links and resources). We end up with a "pseudo" global passive adversary while still managing to only target a vulnerability *within* the Tor threat model.We will not consider any other types of Tor attacks such as compromising routers or keys, replay attack, and directory information spoofing attacks.

[3]Our technique works well only when we use a well provisioned probing node. This means that the bottleneck capacity between the probing node and one of the ORs is at the OR side (including the client to OR link). However, we posit that nodes in major ISPs, government organizations and universities do have such capabilities.

4 Approximating the GPA

The attack on Tor leverages *LinkWidth*, a novel single-end bandwidth-estimation technique that we developed, and works in three phases:

- An adversary continuously senses the available bandwidth in the up-links of all Tor relays. These may be the immediate up-link, or some other link that carries all traffic to/from the Tor node.
- When an anonymous user contacts a server of interest and requests data (*e.g.,* a web page), the traffic from the server to the Tor exit node is artificially modulated. This modulation can be done by the server itself, or by an upstream router that is under the control of the attacker. Alternatively, this modulation may also be induced by a client who intends to unveil the identity of the Tor relays and hidden services. The modulation can be as simple as temporarily queuing all traffic and then releasing it in a high-volume burst, or may involve a unique throughput pattern. The goal of the adversary is to detect this pattern with high confidence as it manifests itself in three Tor relays and underlying router. (Tor, by default, uses nodes in each circuit; our scheme is not sensitive or limited to this number, as we show in Section 5.1.)

Alternatively, an attacker interested in identifying *all* users accessing a server that is not under his control may launch a network denial of service attack against the server or one of its up-links, causing a back-off in TCP connections and hence an increase in available bandwidth in these links traversed by those connections. This scenario requires more resources (in terms of bandwidth) on the part of the attacker. Lacking these, we decided to focus on the malicious/compromised server/router scenario. We note that we require *at most* one such router, and its identity/location is independent from that of the client.

- Once the Tor relays in a circuit are identified, the attacker begins a trace-back process anchored on these nodes (excluding the exit node). Using a pre-established map of the network, the attacker tries to detect further induced traffic fluctuations, one link at a time until an end-host is reached. The same method can be employed to uncover the identity of a Hidden Service. Assuming a client under control, the adversary varies the available bandwidth of the client-server connection following the fluctuation, link-by-link until the entire path from an Entry Node to the Hidden Server via the Tor network is exposed.

Since LinkWidth is a key and novel component of our attack, we devote the remainder of this section to its description. We begin by defining and describing bandwidth estimation techniques and relevant terms;

and thereby describe how LinkWidth estimates available bandwidth. We examine the effectiveness of our attack in the next section.

4.1 Bandwidth Estimation Techniques

Bandwidth is a very broad and sometime ambiguous term. Throughout Computer Science and Electrical Engineering literature, the term bandwidth has been used to quantify different network characteristics. We define bandwidth simply to be the number of bits transferred using IP packets per unit time. However, to further avoid confusion, for the remainder of this paper, we use the more accurate terms *Capacity* and *Throughput*. Any usage of the term bandwidth shall refer to Capacity.

Capacity refers to the maximum possible bits transferred (in the form of IP packet payload) per unit time, through a link or multi-hop path. In the case of a multi-hop path, the capacity is the maximum possible bits transferred per unit time by the *bottleneck* (*i.e.,* "slowest") link. This link may be a bottleneck either due to congestion or due to inherent medium and network device characteristics.

Throughput refers to the maximum possible bits transferred successfully (in the form of IP packet payload) per unit time through a link or multi-hop path. For throughput, we count how many bits are successfully received by the receiver. This differs from capacity because it depends on the ordering of the packets. To measure throughput, protocols such as TCP measure the time dispersion between consecutive acknowledgments, in predefined intervals (usually, every RTT seconds). Thus, throughput is the number of bits correctly received since the correct reception of the last acknowledgment (measured in bits per unit time). One may view capacity/available bandwidth as an "instantaneous" value of throughput. The measurement is hence based on the reception of each packet and may vary over time. There is actually no clear distinction between throughput and capacity other than the timescale involved in their measurement.

4.2 TCP Westwood Congestion Control

In Section 2, we mentioned that LinkWidth uses the TCP Westwood Congestion Control mechanism to perform throughput and capacity estimation. In the next couple of paragraphs we shall describe briefly why congestion occurs in networks and how it is mitigated (in particular, how TCP alleviates congestion). We shall then describe how TCP Westwood Congestion Control works and how LinkWidth uses its principles to track capacity/throughput changes in network paths and links. Readers familiar with these details can skip ahead to Section 4.3.

Network congestion occurs due to lack of capacity in routers or host to "immediately" forward traffic. Due to excessive packets being served by a router per unit time, it reaches the threshold of its processing and forwarding capacity. This results in queuing delays and packet loss, leading to degradation in quality of service. Network congestion control is a very complex area of research since it involves congestion control being implemented at various levels in the stack, including networking devices in the core of the network as well as the transport protocols at end-hosts. We focus on congestion control done by TCP at the end-hosts, and TCP Westwood in particular.

Traditional TCP, commonly known as TCP Tahoe [27], uses a window-based congestion control. TCP ensures reliable delivery of bytes using acknowledgment (ACK) packets. Congestion window, (*cwin*), specifies the number of packets (mostly carrying "Maximum Segment Size" bytes of TCP payload) which the sender sends without caring about an ACK. A correct ACK following correct reception of *cwin* packets assures the sender that there is enough capacity for the routers in the path to correctly forward its packet. Three duplicate ACKs for the previous acknowledged packet or absence of any ACKs for a predefined timeout indicates congestion. Depending up the value of *Slow Start Threshold (ssthresh)*, the value of *cwin* is adjusted with a hope to recover from the perceived network congestion.

Traditionally TCP decreases the *ssthresh* to half the *cwin* when congestion is detected. This may possibly lead to bandwidth under-utilization. Newer TCP variants such as TCP Westwood and TCP Westwood+, geared towards full available bandwidth utilization, suggest sampling the throughput every RTT seconds and using it when adjusting *ssthresh* (thus avoiding under-utilization). *The other advantage of these techniques is that they are "sender-only" modifications (and can work with any TCP receiver).* The throughput is estimated by measuring the time dispersion between two consecutive ACKs and is given by the expression $b_k = L/(t_n - t_{n-1})$. b_k is the measured "instantaneous" bandwidth (measured throughput), L is the length of the payload successfully sent (in bits) between the n^{th} and $(n-1)^{th}$ ACKs, and t_n and t_{n-1} are the time of reception of the n^{th} and $(n-1)^{th}$ ACKs respectively.

TCP Westwood also suggests a metric of *Available Bandwidth Estimate* (BWE). The value of BWE is the weighted average of the most recently measured b_k and the previous one, b_{k_p}. BWE is computed as:

$$BWE_i = (\alpha) * BWE_{i-1} + (1 - \alpha) * ((b_k + b_{k_p})/2)$$

Here, BWE_i is the bandwidth estimate or the throughput estimate between the n^{th} and $(n-1)^{th}$ ACKs. BWE_{i-1} is the throughput estimate for the throughput estimate between the $(n-1)^{th}$ and the $(n-2)^{th}$ ACKs. The α parameter controls the weight of the current estimate relative to the computed historical measurement. (A commonly used value of α is 0.5, which gives equal weight to the previous estimate and the moving average of the current and previous measurements.)

The following pseudo-code show how TCP Westwood adjusts *cwin* and *ssthresh* when a congestion is detected by the expiration of a *coarse timeout*.

```
1:  if $cwin < ssthresh$ then
2:      $a \leftarrow a + 1$
3:      if $a > 4$ then
4:          $a \leftarrow 4$
5:      end if
6:  end if
7:  if $cwin \geq ssthresh$ then
8:      $a \leftarrow 1$
9:  end if
10: $ssthresh \leftarrow (BWE * RTT)/(packet\_size * 8 * a)$
11: if $ssthresh > 2$ then
12:     $ssthresh \leftarrow 2$
13:     $cwin \leftarrow 1$
14: end if
```

The a parameter, *packet_size* and Bandwidth Estimate (*BWE*) are used in adjusting the slow start threshold when congestion is detected.

4.3 LinkWidth : TCP Westwood Sender

We implemented a TCP Westwood sender in LinkWidth, a tool for performing single-end network capacity and link throughput measurements. LinkWidth requires no support or active collaboration from a remote host or any device in the network.

Since the ACK reception can signal correct reception of a packet, LinkWidth sends TCP SYN packets to closed ports. The receiver (a router or end host) replies to such packets with a TCP packet where the RST and ACK flags are set. Where TCP packets are filtered and/or rate limited due to security considerations, we rely on ICMP ECHO_REPLY messages from the receiver to signal correct reception of probe packets (by sending ICMP ECHO_REQUEST packets instead of TCP SYNs). To measure end-to-end TCP capacity, the sender emulates the TCP Westwood sender by sending *cwin* packets. $cwin - 2$ TCP RST packets (called *load packets*), are "sandwiched" between two TCP SYN packets (called the *head measurement packet* and *tail measurement packet* respectively).These TCP SYN packets, sent to closed ports, evoke TCP RST+ACK reply packets destined for the sender of the TCP SYN packets. Figure 3 shows this arrangement of packets. Correct reception of the train of $cwin+1$ packets is determined by two TCP RST+ACK packets from the receiver (for the head and tail measurement packets). Each correct reception of the TCP RST+ACK pair causes *cwin* to be increased either exponentially (Slow Start phase) or linearly (Congestion Avoidance phase). Since we do not rely on a proper TCP connection, the only way to signal a packet loss is by coarse timeout. After sending the train, the sender initiates a timer to wait for the two expected ACKs. The expiry of the timeout causes the readjustment of the *cwin* and *ssthresh* parameters inside a timeout event handler.

We send TCP RST packets is to avoid unnecessary replies, either in the form of TCP RST or ICMP Destination Host/Net Unreachable packets, that can interfere with our forward probe traffic. The time dispersion between two consecutive TCP RST+ACK replies due to the head and tail measurement packets are stored as t_n and t_{n-1}. Thus the capacity/bandwidth is measured as:

$$b_k = (cwin * L)/(t_n - t_{n-1})$$

Here, b_k is the measured "instantaneous" bandwidth (measured throughput), $cwin * L$ is the total data sent (in bits) for the entire train, t_n and t_{n-1} are the times of reception of the two TCP RST+ACK reply packets. Our method is a direct extension of the packet train method. The successful reception to a previous train determines how many packets we send in the current train.

Throughput measurement is a slight modification of the capacity measurement technique. The TCP RST load packets are replaced by TCP SYN packets (all destined to closed ports on the receiver). The time of reception of the TCP RST+ACK due to the first TCP SYN packet is stored in the variable *first*. Figure 4 shows this arrangement of packets. Thus, for any value of *cwin*, if any m replies are received correctly (such that $1 \leq m \leq cwin$), this indicates that the throughput is:

$$b_k = (m * L)/(t_m - first)$$

where t_m is the time when the m^{th} reply is correctly received. LinkWidth reports the measurement as BWE.

In some cases, we observed that TCP SYN packets may be filtered or rate-limited. To counter this, we replace the head and tail TCP SYN packets with ICMP

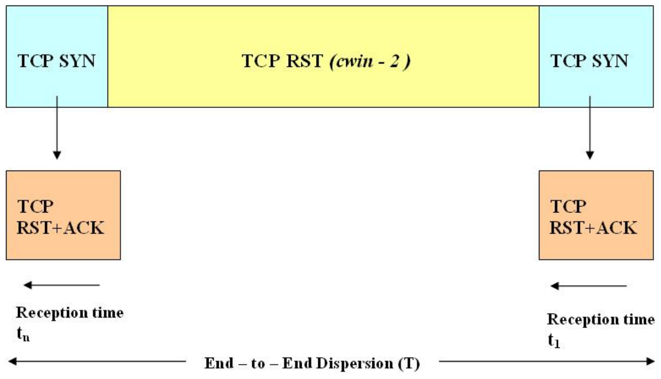

Figure 3. Arrangement of Packets in LinkWidth for Measurement of Capacity

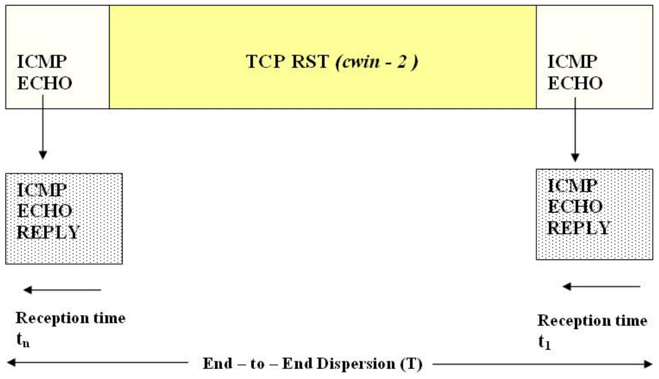

Figure 5. Arrangement of Packets in LinkWidth for Measurement of Capacity (using ICMP)

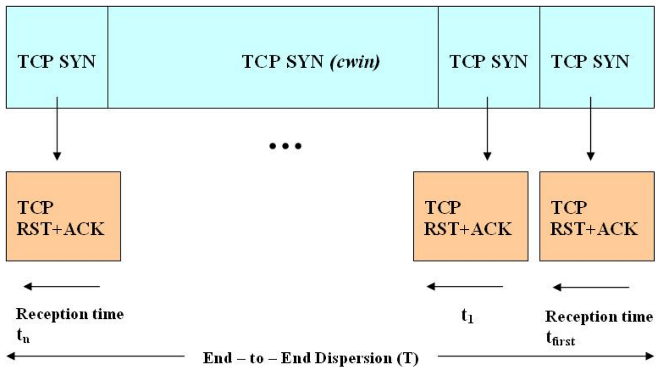

Figure 4. Arrangement of Packets in LinkWidth for Measurement of Throughput

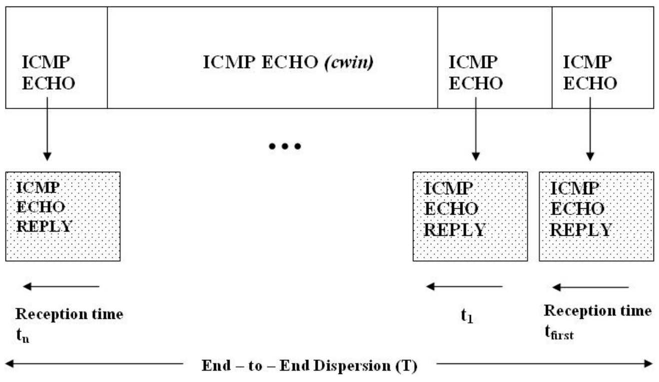

Figure 6. Arrangement of Packets in LinkWidth for Measurement of Throughput (using ICMP)

ECHO packets. The load packets continue to be TCP RST packets. Correct reception of the train is indicated by reception of ICMP ECHO_REPLY packets at the sender. The arrangement of packets is shown in Figure 5. A similar modification is used for measuring throughput: the receiver waits to see how many ICMP ECHO_REPLY response packets it receives before estimating the throughput. The corresponding packet arrangement is shown in Figure 6.

LinkWidth Implementation : We developed a prototype of LinkWidth for GNU/Linux. To avoid incurring packet delays due to kernel resource scheduling, we bypassed the regular protocol stack and send our own TCP and ICMP packets crafted using the Raw Socket API. The coarse timeout is implemented using the standard POSIX API function *setitimer()*. The expiry of the timer is indicated by raising a *SIGALRM* signal.

5 Experimental Evaluation

We start by quantifying the effectiveness of LinkWidth in terms of speed and accuracy in detecting traffic variations. To that end, we use various scenarios that involve both cross-traffic and traffic-shaping of links. Our aim is to successfully measure variations of available capacity using LinkWidth using a single host (*i.e.,* without a conveniently placed collaborating peer node, which would severely restrict the flexibility and power of our attack). Moreover, we discuss the role of the different parameters used by the tool. However, since our focus is the detection of traffic variations, we are not going to present the full optimality results for

the tool parameters.
Measuring Capacity: We illustrate our lab testbed in Figure 7. We used *wget* to generate client HTTP request traffic. The server runs an Apache process waiting for client connections. The client(s) connect to the server through Linux hosts (R1 and R2) that are set to forwarding mode (acting as routers). The measuring (probe) host's packets are routed through R2. We configure all of the machines to use static routes. The server shapes the clients' HTTP traffic bandwidth using the Linux Traffic Controller [15] with Hierarchical Token Bucket.

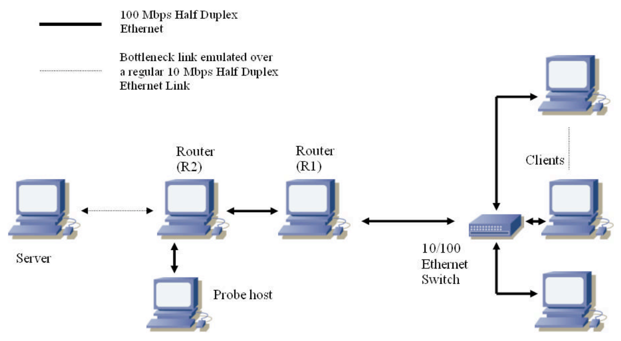
100 Mbps Half Duplex Ethernet
Bottleneck link emulated over a regular 10 Mbps Half Duplex Ethernet Link
Router (R2)
Router (R1)
Clients
Server
Probe host
10/100 Ethernet Switch

Figure 7. **Lab testbed for measuring available capacity**

In the remainder of this section, we establish the limits of our measuring methodology which extends to traffic variations. As we will show, LinkWidth can observe 1 Kbps of un-elastic UDP traffic and detect variation of 50 Kbps of TCP traffic with very high confidence (> 95%). We begin by measuring variations observed for available link capacity for elastic TCP traffic flows (HTTP) in the presence of cross-traffic.

Capacity Measurement in the Presence of Competing TCP Traffic In this scenario, three clients are connected to a web server. Each client initiates a *wget* request to download a 100 MByte file from the server. We use LinkWidth to measure the change in available bandwidth due to each client closing its connection in succession. The choice of a relatively large file allows us to sustain traffic for a sufficiently long window of time. As we will show, this is not a hard requirement and help us establish a measurement baseline. The measuring host probes for the available capacity of the link connecting $R2$ to $R1$. $R2$ employs Nistnet [5] to emulate a 10 Mbps half-duplex link over a regular 100 Mbps half-duplex Ethernet link. The server uses traffic shaping to limit each HTTP connection to a maximum of 500 Kbps. The results of this experiment are tabulated in Table-1.

Number of Clients	**Available Capacity Average(bps)**	**Median(bps)**
3	7,372,525.47	7,518,356.26
2	8,119,550.66	8,150,574.70
1	8,618,681.63	8,600,065.92
No clients	9,300,960.91	9300960.91

Table 1. **Increasing HTTP traffic resulting in decreasing available capacity, as measured by LinkWidth**

For these experiments, we choose probes with packet sizes 1400 bytes to maximize link utilization (the Ethernet MTU being 1500 bytes). The α parameter, described previously, is set to 0.5 and the inter-probe delay is 100 milliseconds.

In the absence of any traffic, the 10 Mbps link reports a capacity of approximately 9.3 Mbps. When all the three hosts download simultaneously, we measure a total throughput of approximately 7.4 Mbps. This is as expected (as all of the 500Kbps connections together achieve 1.5 Mbps).

This experiment demonstrates the effectiveness of LinkWidth in measuring the available capacity by quickly adjusting to the dynamic TCP cross-traffic. The reported available capacity increases when the connections are closed one by one in quick succession. However, we are yet to present evidence of LinkWidth's accuracy and granularity.

Measurement Using Small Files The next experiment provides such evidence. We continue to use the previous experimental setup, shown in Figure 7. The server shapes the available capacity of a single client connection. The client downloads a relatively small file (2 MBytes) from the server. A small-sized file is chosen to demonstrate the speed with which LinkWidth converges to the value of available capacity. The rest of the experimental setup and parameters are the same as in the previous experiment. The adversary probes the bottleneck link for each of the various available capacity levels. The results of this probing are shown in through a histogram in Figure 8.

The accuracy and granularity are highlighted by these results. The upper limit of HTTP traffic is increased from 200 Kbps to 1.4 Mbps in increments of 200 Kbps. Evident from the results, there is a definite noticeable decrease in the the available capacity for each increment. Similar results are obtained with a 50 Kbps increment. Once again, the probe packet size, the α parameter and the inter probe latency are the same as in the previous experiment.

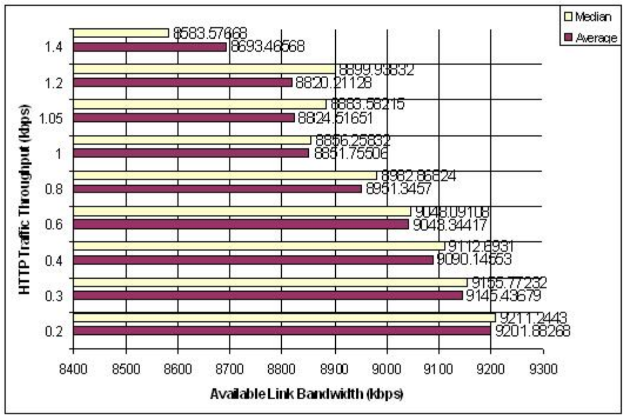

Figure 8. **Increasing HTTP traffic resulting in decreasing available capacity, as measured by LinkWidth**

We next describe the experiments that demonstrate the effectiveness of our technique in detecting the fluctuations in capacity of the relays and routers involved in a Tor circuit.

5.1 Traffic Analysis Against Tor

We use LinkWidth to detect induced traffic fluctuations in Tor relays participating in a circuit and on routers connecting a Tor Entry Node and the Tor clients and Hidden Servers. We demonstrate complete trace-back attack linking a Tor Clients (OPs) and Hidden Server to their Entry Nodes achieved by measuring the fluctuation of available capacity/throughput to routers involved in the circuit between the OP and the Entry Node. We begin by demonstrating how a GPA can use LinkWidth in determining the Tor relays involved in a circuit. Thereby we show how Tor may be used for determining the identity of anonymized clients and servers.

Probing Tor Relays

This subsection focuses on demonstrating how LinkWidth maybe used for performing traffic analysis against Tor relays (ORs) participating in a Tor circuit. The next few paragraphs describe the set-up used for such the attack. Thereafter, we describe our attack and conclude by presenting the effectiveness our technique through results from attacking a small subset of all possible Tor circuits.

Probe Set-up & Technique for Identifying Tor Relays : Figure 9 illustrates how the adversary probes the Tor relays involved in a circuit. In our experiments, we use LinkWidth to probe Tor nodes (ORs) that may possibly be part of Tor circuits. An adversary with sufficient bandwidth resources would be simultaneously probing all (or a large portion of) Tor nodes. Patterns in the traffic are introduced by shaping these circuits at the server. *The goal of the experiment is to demonstrate that an adversary is able to detect these induced fluctuations in bandwidth in the Tor relays (participating in the circuit) whenever the client downloads the file from the server.*

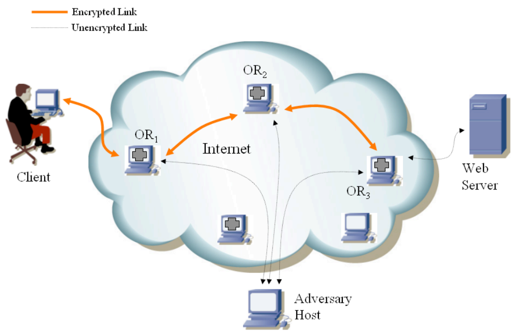

Figure 9. **Adversary probing the fluctuation in available bandwidth of ORs participating in a Tor circuit**

To bypass the default restrictions on Middleman Node selection, we modified the Tor Client *version 0.1.2.18* to enable the establishment of circuits where the user can select all the ORs manually[4].

In our experiments, the client, the web server and the probing host used by the adversary are all scattered in separate geographic locations and networks within the US[5]. The web server offers a 100 MByte file, a relatively large file for providing adequate delay to the adversary for perceiving the changing network congestion at each value of the client-server traffic bandwidth. The web server controls the available capacity of the client-server connection by shaping the web traffic using the Linux Traffic Controller [15].

The client selects the relays to be used in the circuit from the frequently updated and publicly viewable Tor Status page [18]. The adversary probes the Tor relays in the circuit to detect traffic variations whenever

[4]Selection of the intermediate relay nodes cannot be controlled in the standard Tor distribution. There are, however, source-code options that can unlock the intermediate relay selection.

[5]We avoid selecting nodes in Europe and Asia so as to avoid inter-continental Internet links which may at times act as bottlenecks. An adversary would need nodes in these geographical locations from which to probe the relevant Tor relays. While this is generally feasible, we do not currently have such capabilities.

the client communicates to the web server through the circuit. The client downloads a file from the server through Tor. The server colludes with the adversary to limit the web traffic to various bandwidth levels.

We quantify the effectiveness of detecting the ORs involved in Tor circuits by creating 50 distinct Tor circuits[6]. We probed these relays from different network locations. The results summarized in Table 3 show how many of the Tor relays in each circuit reported fluctuation in available bandwidth (and were thus correctly identified by the attacker).

Relays/Circuit Detected	**# of Circuits**
3	11
2	14
1	12
0	13

Table 2. Number of Tor relays per Tor circuit where available bandwidth fluctuation is correctly detected.

In our experiments, 11 of the 50 circuits that were probed, revealed correct variation in bandwidth for all the relays involved, while in 14 circuits we were able to identify only 2 of the 3 relays involved. There were 13 circuits in which only one 1 of the relays was detected. Finally, we also have 12 circuits where we are not able to detect any of the participating ORs. Among all the 150 ORs probed there were 22 which filtered all probe traffic.

These results are in effect the true-positives/false-negatives values. Thus, evident from table 3, the truepositive rate is:

Total number of nodes probed = 128 (N) (not counting the 22 which filtered all probe traffic)
Total number of nodes in which bandwidth fluctuation was observed = 74 (T)
True Positives (T/N) = 74/128 (57.8%)
False Negatives ($1 - T/N$) = 54/128 (42.2%)

The same experimental setup was used to determine the false-positives. For this, we created 10 different regular "3-hop" Tor circuits. We selected ORs which were not participating in any of the circuits. The results from these experiments are summarized as follows.

Total number of nodes probed = 30 (N)
Total number of non-participating nodes which report fluctuation = 3 (F)
False Positives (F/N) = 3/30 (10%)
True Negatives ($1 - F/N$) = 27/30 (90%)

In the first attempt we observed bandwidth fluctuation on relays which were not part of our client's anonymous circuit. However on repeated attempts of the same experiment we saw no fluctuation in available bandwidth; thereby detected no false positives.

We quantify the effectiveness of detecting the ORs involved in Tor circuits by creating 50 distinct Tor circuits[7]. We probed these relays from different network locations. The results summarized in Table 3 show how many of the Tor relays in each circuit reported fluctuation in available bandwidth (and were thus correctly identified by the attacker).

Relays/Circuit Detected	**# of Circuits**
3	11
2	14
1	12
0	13

Table 3. Number of Tor relays per Tor circuit where available bandwidth fluctuation is correctly detected.

In our experiments, 11 of the 50 circuits that were probed, revealed correct variation in bandwidth for all the relays involved, while in 14 circuits we were able to identify only 2 of the 3 relays involved. There were 13 circuits in which only one 1 of the relays was detected. Finally, we also have 12 circuits where we are not able to detect any of the participating ORs. Among all the 150 ORs probed there were 22 which filtered all probe traffic.

These results are in effect the true-positives/false-negatives values. Thus, evident from table 3, the truepositive rate is:

Total number of nodes probed = 128 (N) (not counting the 22 which filtered all probe traffic)
Total number of nodes in which bandwidth fluctuation was observed = 74 (T)
True Positives (T/N) = 74/128 (57.8%)
False Negatives ($1 - T/N$) = 54/128 (42.2%)

[6]Recall, this is the first step in a follow-on attack that identifies the end-user.

[7]Recall, this is the first step in a follow-on attack that identifies the end-user.

The same experimental setup was used to determine the false-positives. For this, we created 10 different regular "3-hop" Tor circuits. We selected ORs which were not participating in any of the circuits. The results from these experiments are summarized as follows.

Total number of nodes probed = 30 (N)
Total number of non-participating nodes which report fluctuation = 3 (F)
False Positives (F/N) = 3/30 (10%)
True Negatives ($1 - F/N$) = 27/30 (90%)

In the first attempt we observed bandwidth fluctuation on relays which were not part of our client's anonymous circuit. However on repeated attempts of the same experiment we saw no fluctuation in available bandwidth; thereby detected no false positives.

Most of the ORs filter and/or rate limit TCP SYN packets to closed ports. In the presence of such filtering, we use the ICMP-based emulation of LinkWidth. Probes using ICMP are prone to error due to difference in dispersion of the replies from those of the forward probe traffic (when the probes reach the destination).

From our experiments, it appears that correct detection of the exact pattern of increasing or decreasing available capacity is contingent upon various factors. The most restricting is whether the adversary is at a network "vantage" point. This simply means that the bottleneck of the path from the adversary to the relay is the network interface of the relay. If this condition does not hold, then accurate bandwidth measurement may not be possible in all cases (depending on link utilization). In practice, our use of a few vantage points in academic institutions in the US seems to provide sufficient bandwidth to conduct our attack. For an attack against all of Tor, we would require access to nodes in the different major geographical areas hosting Tor relays (Europe, Asia) such that we do not have to probe over trans-Atlantic or trans-Pacific long-haul links that affect the accuracy of our measurements.

Detecting a "6-hop" Tor Circuit. We use our modified Tor client program to create a longer Tor circuit, in an attempt to hide participating ORs through expected attenuation of the pattern as it traverses more links, routers, and Tor nodes. We select six relays which were correctly detected in the previous experiment and created a circuit involving these. We repeated the same experiment with the same client and server. The client successfully achieves approximately 80 KBytes/sec. We are able to effectively observe bandwidth fluctuations in all six relays. We believe we can observe traffic fluctuation with higher number of hops, provided that the client achieves at least 30–40 KBytes/sec throughput. We conducted a 10-hop Tor circuit experiment. However due to the present lack of functional exit relays within the US that provide high throughput rates to the client, we were unable to complete the experiment — the capacity of the circuit was simply too low (and unsuitable for practical use).

Identifying Tor Clients & Hidden Services

In the previous subsection 5.1 we presented results from probing Tor relays participating in Tor circuits. In this subsection we present experiments for probing routers connecting Tor clients and Tor Hidden Services to their Entry Nodes . The experimental setup is very similar to that in the previous subsection.

The attempt in all experiments throughout this subsection is to utilize network bandwidth information along the path from the client or the server/Hidden Server to its Entry Node to expose its identity and thus demonstrate the effectiveness of our technique.

Probe Set-up & Technique for Identifying Tor Clients : Figure 10 explains how an adversary uses LinkWidth to probe all possible routers along network paths connecting the Tor Entry Node to the Tor Client. As earlier, the client fetches a relatively large file from the server, which shapes the bandwidth of the connection. The adversary tries to determine the fluctuations introduced by the server along the routers connecting the client to the Entry Node. Since we don't have a map of link-by-link connectivity of the Internet, we rely on traceroute information for determining the hops between the client and its Entry Node.

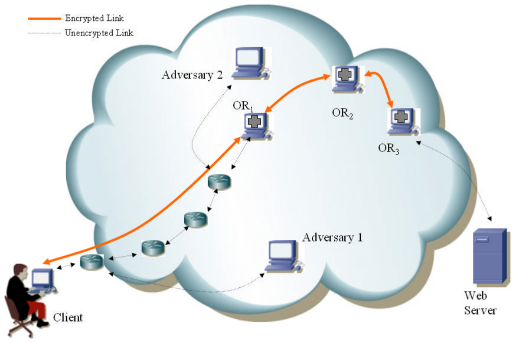

Figure 10. The adversary traces bandwidth fluctuation between the Tor client and the Tor Entry Node, one network link at a time

The results from probing routers connecting an OP to an Entry Node are presented in table 4 the Appendix.

We probed ten separate Tor Client – server circuits. In all of the cases the adversary could successfully detect the bandwidth fluctuation only in some of the routers. This is partially due to insufficient number of existing vantage points and partly because because some of the routers were unresponsive to the probes. In some cases, the client traffic achieved throughput less than 10 KBytes/sec (approximately 82 Kbps, which is less than 30–40 KBytes/sec, necessary for detecting available bandwidth fluctuation on Internet paths, when probed using LinkWidth). In a crude sense, the reader may consider this an inherent measurement granularity of LinkWidth.

Correct detection of bandwidth fluctuation is achieved through a perceived change in packet loss whenever the routers and/or relays are probed using LinkWidth; while the client downloads the from the colluding server that shapes the available bandwidth of the HTTP connection.

Probe Set-up & Technique for Identifying Tor Hidden Servers : The set-up for determining the identity of Hidden Services, though similar to the one used to identifying Tor clients differ slightly. It is shown in Figure 11. Here the adversary (or adversaries), probe the routers connecting routers connecting the Hidden Services to their Entry Nodes. This is used to to unveil the identity of a Hidden Service. *In these experiments, the available bandwidth fluctuation was induced by the client.* As earlier, rely solely on traceroute for determining which routers connect a Hidden Server to its corresponding Entry Node.

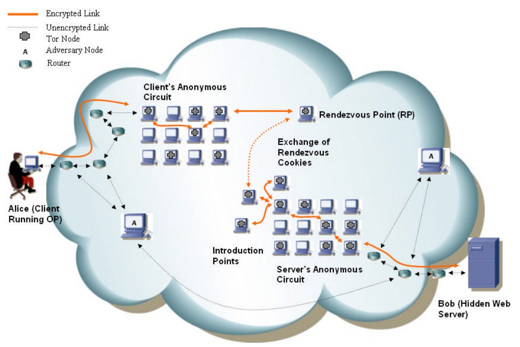

Figure 11. The adversary is tracing bandwidth fluctuation between the anonymous client / Hidden Server and the Tor entry node, one network link at a time

We probed the paths of ten separate Tor client–Hidden Server circuits. Specifically, The adversaries (probe hosts) probed the available bandwidth fluctuation in the routers connecting the Hidden Servers to their corresponding Entry Nodes. Table 5 in the Appendix summarizes the results of this experiment.

In almost every circuit, there were some routers which filtered our probed packets. The rest were either detected correctly or not detected; due to insufficient existing vantage points and insufficient throughput in client's traffic (approximately 5–10 KBytes/sec).

6 Discussion

In all the attacks presented so far, we avoided crossing the continent in our search for relays. The transcontinental links are a bottleneck in many instances. Moreover, we are only able to accurately probe hosts for which our probing hosts are at a vantage location in terms of bandwidth [8]. Unfortunately, PlanetLab [20] hosts cannot provide accurate traffic measurements because of the absence of support for the ability to control kernel scheduling of network resources. Intermittent quality of service is the other major factor that seriously affects our attempt in probing relays participating in Tor circuits.

To obtain high confidence detection results, we need to observe traffic from Tor nodes with throughput of at least 30–40 KBytes/sec (approximately 300 Kbps). In addition, packet loss, traffic filtering and shaping, intermediate network bottlenecks and operating system and/or networking device driver dependent issues play an important role in measurement based network monitoring. PlanetLab hosts are a good case in point of how different parameters can significantly impact measurement fidelity. LinkWidth's TCP probes are rate limited/filtered when run on PlanetLab hosts. ICMP replies from the ICMP probes arrive within a very short time interval, resulting in an inaccurate estimation of the received dispersion of the probe train. Such filtering is also observed in some Tor relays that are probed from the adversary at one of the vantage points used in our experiments. In most instances where TCP probes were rate-limited or even filtered, we rely on ICMP probes.

The effectiveness of the Tor attacks presented earlier is constrained by the limitations of the traffic-based measuring techniques. These limitations could be leveraged to create countermeasures to our attack. First, a Tor client can use parallel circuits in a round-robin fashion to access the same server; this would diffuse the ability of the server to generate detectable traffic variations, since traffic spikes would be distributed across all the parallel connections. The use of shorter circuit lifetimes would make it more difficult for an adversary to correctly traceback from detected Tor nodes

[8]We currently do not have access to such hosts abroad

through the network. Depending on the size of the Tor network, the resources of the adversary, and the circuit lifetime, we could (as part of future work) analytically estimate or simulate the probability of full or partial circuit and client exposure over multiple interactions with the server. Traffic smoothing by Tor relays is another potential partial countermeasure.

7 Conclusion

We propose a new technique for uncovering Tor relays, tracing back to the client or to a Hidden Service using a novel single-end bandwidth estimation technique. Our scheme works by artificially inducing traffic fluctuations to the traffic sent by a server (or a client) to the anonymous client (or to a Hidden Service). This is achieved by colluding with the server (or client), controlling an upstream router, or through a DDoS attack. By detecting these perturbations as they traverse the network using single-end bandwidth measurements, an attacker effectively acts as a global passive adversary. For accurately detecting the relevant links, it is essential for an adversary to be at a "vantage point" in the network such that either the bottleneck is the link itself or the disturbance in cross traffic caused by the server is enough to distort LinkWidth's probes.

References

[1] D. Agrawal and D. Kesdogan. Measuring Anonymity: The Disclosure Attack. *IEEE Security & Privacy*, 1(6):27–34, November/December 2003.

[2] D. Antoniades, M. Athanatos, A. Papadogiannakis, E. P. Markatos, and C. Dovrolis. Available Bandwidth Measurement as Simple as Running wget. In *Proceedings of Passive and Active Measurements (PAM)*, March 2006.

[3] K. Borders and A. Prakash. Web Tap: Detecting Covert Web Traffic. In *Proceedings of the* 11^{th} *ACM Conference on Computer and Communications Security (CCS)*, pages 110–120, October 2004.

[4] H. Burch and B. Cheswick. Tracing Anonymous Packets to Their Approximate Source. In *Proceedings of the* 14^{th} *USENIX LISA Conference*, pages 319–327, December 2000.

[5] M. Carson and D. Santay. NISTNet-A Linux-based Network Emulation Tool. `http://www-x.antd.nist.gov/nistnet/nistnet.pdf`.

[6] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *In Proceedings of the* 13^{th} *USENIX Security Symposium*, pages 303–319, August 2004.

[7] C. Dovrolis and R. Prasad. Pathrate. `http://www.cc.gatech.edu/fac/Constantinos.Dovrolis/pathrate.tar.gz`, 2004.

[8] A. B. Downey. Using pathchar to Estimate Internet Link Characteristics. In *Proceedings of ACM SIGCOMM*, August 1999.

[9] M. J. Freedman and R. Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002)*, Washington, DC, November 2002.

[10] M. Gerla, M. Y. Sanadidi, R. Wang, and A. Zanella. TCP Westwood: Congestion Window Control Using Bandwidth Estimation. In *Proceedings of IEEE Globecom, Volume 3*, pages 1698–1702, November 2001.

[11] L. A. Grieco and S. Mascolo. Performance evaluation and comparison of Westwood+, New Reno and Vegas TCP congestion control. *ACM Computer Communication Review*, 34(2), April 2004.

[12] Y. Guan, X. Fu, D. Xuan, P. Shenoy, R. Bettati, and W. Zhao. Efficient Traffic Camouflaging in Mission-Critical QoS-Guaranteed Networks. *IEEE Transactions on Systems, Man, and Cybernetics*, 31, July 2001.

[13] B. Hajek and B. Radosavljevic. Hiding Traffic Flow in Communication Networks. In *Proceedings of the IEEE Military Communication Conference (MilCom)*, October 1992.

[14] N. Hopper, E. Y. Vasserman, and E. Chan-Tin. How Much Anonymity does Network Latency Leak? In *Proceedings of ACM CCS*, October 2007.

[15] B. Hubert, T. Graf, G. Maxwell, R. Mook, M.Oosterhout, P.Schroeder, J. Spaans, and P. Larroy. Linux Advanced Routing and Traffic Control HOWTO. `http://lartc.org/howto`.

[16] V. Jacobson. PATHCHAR. `http://www.caida.org/tools/utilities/others/pathchar/`, 1997.

[17] S. Keshav. Congestion Control in Computer Networks. UC Berkely Technical Report TR-654, September 1991.

[18] J. B. Kowalski. TorStatus. `http://anonymizer.blutmagie.de:2505/`.

[19] S. J. Murdoch and G. Danezis. Low-Cost Traffic Analysis of Tor. In *IEEE Symposium on Security and Privacy*, pages 183–195, May 2005.

[20] PlanetLab. `http://www.planet-lab.org/`.

[21] R. Prasad, M.Murray, C. Dovrolis, and K. Claffy. Bandwidth Estimation: Metrics, Measurement Techniques, and Tools. In *Proceedings of IEEE Network*, August 2003.

[22] J.-F. Raymond. Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems. In *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 10–29. Springer-Verlag, LNCS 2009, July 2000.

[23] J.-F. Raymond. Traffic Analysis: Protocols, Attacks, Design Issues and Open Problems. In *Proceedings of the International Workshop on Design Issues in Anonymity and Unobservability*, pages 10–29, 2001.

[24] M. Y. Sanadidi. Bandwidth Estimation Techniques , A Tutorial Presentation. In *SBRC 2002:Brazilian Symposium on Computer Networks Date*, Buzios, Brazil, May 2002.

[25] A. Serjantov and P. Sewell. Passive Attack Analysis for Connection-Based Anonymity Systems. In *Proceedings of ESORICS*, October 2003.

[26] Stefan, Saroiu, and Krishna. Sprobe: Another tool for measuring bottleneck bandwidth. In *Proceedings of InfoComm 2002*, 2002.

[27] W. Stevens. RFC 2001 - TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. `http://www.faqs.org/rfcs/rfc2581.html`, 1999.

[28] P. F. Syverson, D. M. Goldschlag, and M. G. Reed. Anonymous Connections and Onion Routing. In *IEEE Symposium on Security and Privacy*, pages 44–54, May 1997.

[29] A. Tirumala, F. Qin, J. Dugan, J. Feguson, and K. Gibbs. IPERF. `http://dast.nlanr.net/projects/Iperf/`, 1997.

[30] B. R. Venkatraman and R. E. Newman-Wolfe. High Level Prevention of Traffic Analysis. In *Proceedings of the* 7th *Annual Computer Security and Applications Conference (ACSAC)*, December 1991.

[31] X. Wang and D. S. Reeves. Robust Correlation of Encrypted Attack Traffic Through Stepping Stones by Manipulation of Interpacket Delays. In *Proceedings of the* 10th *ACM Conference on Computer and Communications Security (CCS)*, pages 20–29, October 2003.

APPENDIX

Circuit Number	# of hops from Client–Entry Node	Correctly Detected Client–Entry Node Hops	Unresponsive Routers	Routers Not Reporting Enough Fluctuation	Success Rate
1	10	6	4	0	60.00%
2	15	4	0	0	26.67%
3	18	4	7	12	22.23%
4	18	5	8	5	27.78%
5	14	6	2	6	42.86%
6	14	9	1	4	64.30%
7	15	7	2	6	46.67%
8	14	7	2	5	50.00%
9	14	4	2	8	28.57%
10	15	6	4	5	40.00%

Table 4. Available-bandwidth fluctuation detection in links connecting a Tor client and to its Entry Node

Circuit Number	# of hops from Server–Entry Node	Correctly Detected Server–Entry Node Hops	Unres-ponsive Routers	Routers Not Reporting Enough Fluctu-ation	Success Rate
1	13	4	2	7	30.70%
2	12	9	0	3	75.00%
3	11	7	1	3	63.64%
4	14	5	4	5	35.71%
5	12	9	0	3	75.00%
6	13	3	3	7	23.08%
7	16	5	5	6	31.25%
8	13	3	2	8	23.08%
9	17	4	1	12	23.53%
10	13	5	1	7	38.46%

Table 5. Available-bandwidth fluctuation detection in links connecting a Hidden server to its Entry Nodes