

# A Practical Congestion Attack on Tor Using Long Paths

Nathan S. Evans  
*Colorado Research Institute  
for Security and Privacy  
University of Denver  
Email: nevans6@du.edu*

Roger Dingledine  
*The Tor Project  
Email: arma@mit.edu*

Christian Grothoff  
*Colorado Research Institute  
for Security and Privacy  
University of Denver  
Email: christian@grothoff.org*

## Abstract

In 2005, Murdoch and Danezis demonstrated the first practical congestion attack against a deployed anonymity network. They could identify which relays were on a target Tor user's path by building paths one at a time through every Tor relay and introducing congestion. However, the original attack was performed on only 13 Tor relays on the nascent and lightly loaded Tor network.

We show that the attack from their paper is no longer practical on today's 1500-relay heavily loaded Tor network. The attack doesn't scale because a) the attacker needs a tremendous amount of bandwidth to measure enough relays during the attack window, and b) there are too many false positives now that many other users are adding congestion at the same time as the attacks.

We then strengthen the original congestion attack by combining it with a novel bandwidth amplification attack based on a flaw in the Tor design that lets us build long circuits that loop back on themselves. We show that this new combination attack is practical and effective by demonstrating a working attack on today's deployed Tor network. By coming up with a model to better understand Tor's routing behavior under congestion, we further provide a statistical analysis characterizing how effective our attack is in each case.

## 1 Introduction

This paper presents an attack which exploits a weakness in Tor's circuit construction protocol to implement an improved variant of Murdoch and Danezis's congestion attack [26, 27]. Tor [12] is an anonymizing peer-to-peer network that provides users with the ability to establish low-latency TCP tunnels, called circuits, through a network of relays provided by the peers in the network. In 2005, Murdoch and Danezis were able to determine the path that messages take through the Tor network by causing congestion in the network and then observing the changes in the traffic patterns.

While Murdoch and Danezis's work popularized the idea proposed in [1] of an adversary perturbing traffic patterns of a low-latency network to deanonymize its users, the original attack no longer works on the modern Tor network. In a network with thousands of relays, too many relays share similar latency characteristics and the amount of congestion that was detectable in 2005 is no longer significant; thus, the traffic of a single normal user does not leave an easily distinguishable signature in the significantly larger volume of data routed by today's Tor network.

We address the original attack's weaknesses by combining JavaScript injection with a selective and asymmetric denial-of-service (DoS) attack to obtain specific information about the path selected by the victim. As a result, we are able to identify the entire path for a user of today's Tor network. Because our attack magnifies the congestion effects of the original attack, it requires little bandwidth on the part of the attacker. We also provide an improved method for evaluating the statistical significance of the obtained data, based on Tor's message scheduling algorithm. As a result, we are not only able to determine which relays make up the circuit with high probability, we can also quantify the extent to which the attack succeeds. This paper presents the attack and experimental results obtained from the actual Tor network.

We propose some non-trivial modifications to the current Tor protocol and implementation which would raise the cost of the attack. However, we emphasize that a full defense against our attack is still not known.

Just as Murdoch and Danezis's work applied to other systems such as MorphMix [24] or Tarzan [36], our improved attack and suggested partial defense can also be generalized to other networks using onion routing. Also, in contrast to previously proposed solutions to congestion attacks [18, 22–24, 28, 30, 35, 36], our proposed modifications do not impact the performance of the anonymizing network.

## 2 Related Work

Chaum’s mixes [3] are a common method for achieving anonymity. Multiple encrypted messages are sent to a mix from different sources and each is forwarded by the mix to its respective destination. Combinations of artificial delays, changes in message order, message batching, uniform message formats (after encryption), and chaining of multiple mixes are used to further mask the correspondence between input and output flows in various variations of the design [5, 7, 8, 17, 21, 25, 32, 33]. Onion routing [16] is essentially the process of using an initiator-selected chain of low-latency mixes for the transmission of encrypted streams of messages in such a way that each mix only knows the previous and the next mix in the chain, thus providing initiator-anonymity even if some of the mixes are controlled by the adversary.

### 2.1 Tor

Tor [12] is a distributed anonymizing network that uses onion routing to provide anonymity for its users. Most Tor users access the Tor network via a local proxy program such as Privoxy [20] to tunnel the HTTP requests of their browser through the Tor network. The goal is to make it difficult for web servers to ascertain the IP address of the browsing user. Tor provides anonymity by utilizing a large number of distributed volunteer-run relays (or routers). The Tor client software retrieves a list of participating relays, randomly chooses some number of them, and creates a circuit (a chain of relays) through the network. The circuit setup involves establishing a session key with each router in the circuit, so that data sent can be encrypted in multiple layers that are peeled off as the data travels through the network. The client encrypts the data once for each relay, and then sends it to the first relay in the circuit; each relay successively peels off one encryption layer and forwards the traffic to the next link in the chain until it reaches the final node, the exit router of the circuit, which sends the traffic out to the destination on the Internet.

Data that passes through the Tor network is packaged into fixed-sized cells, which are queued upon receipt for processing and forwarding. For each circuit that a Tor router is a part of, the router maintains a separate queue and processes these queues in a round-robin fashion. If a queue for a circuit is empty it is skipped. Other than using this fairness scheme, Tor does not intentionally introduce any latency when forwarding cells.

The Tor threat model differs from the usual model for anonymity schemes [12]. The traditional threat model is that of a global passive adversary: one that can observe all traffic on the network between any two links. In contrast, Tor assumes a non-global adversary which can only observe some subset of the connections and

can control only a subset of Tor nodes. Well-known attack strategies such as blending attacks [34] require more powerful attackers than those permitted by Tor’s attacker model. Tor’s model is still valuable, as the resulting design achieves a level of anonymity that is sufficient for many users while providing reasonable performance. Unlike the aforementioned strategies, the adversary used in this paper operates within the limits set by Tor’s attacker model. Specifically, our adversary is simply able to run a Tor exit node and access the Tor network with resources similar to those of a normal Tor user.

### 2.2 Attacks on Tor and other Mixes

Many different attacks on low-latency mix networks and other anonymization schemes exist, and a fair number of these are specifically aimed at the Tor network. These attacks can be broadly grouped into three categories: path selection attacks, passive attacks, and active attacks. Path selection attacks attempt to invalidate the assumption that selecting relays at random will usually result in a safe circuit. Passive attacks are those where the adversary in large part simply observes the network in order to reduce the anonymity of users. Active attacks are those where the adversary uses its resources to modify the behavior of the network; we’ll focus here on a class of active attacks known as congestion or interference attacks.

#### 2.2.1 Path Selection Attacks

Path selection is crucial for the security of Tor users; in order to retain anonymity, the initiator needs to choose a path such that the first and last relay in the circuit won’t collude. By selecting relays at random during circuit creation, it could be assumed that the probability of finding at least one non-malicious relay would increase with longer paths. However, this reasoning ignores the possibility that malicious Tor routers might choose only to facilitate connections with other adversary-controlled relays and discard all other connections [2]; thus the initiator either constructs a fully malicious circuit upon randomly selecting a malicious node, or fails that circuit and tries again. This type of attack suggests that longer circuits do not guarantee stronger anonymity.

A variant of this attack called “packet spinning” [30] attempts to force users to select malicious routers by causing legitimate routers to time out. Here the attacker builds circular paths throughout the Tor network and transmits large amounts of data through those paths in order to keep legitimate relays busy. The attacker then runs another set of (malicious) servers which would eventually be selected by users because of the attacker-generated load on all legitimate mixes. The attack is successful if, as a result, the initiator chooses only malicious servers for its circuit, making deanonymization trivial.

### 2.2.2 Passive Attacks

Several passive attacks on mix systems were proposed by Back et al. [1]. The first of these attacks is a “packet counting” attack, where a global passive adversary simply monitors the initiator’s output to discover the number of packets sent to the first mix, then observes the first mix to watch for the same number of packets going to some other destination. In this way, a global passive adversary could correlate traffic to a specific user. As described by Levine et al. [23], the main method of defeating such attacks is to pad the links between mixes with cover traffic. This defense is costly and may not solve the problem when faced with an active attacker with significant resources; an adversary with enough bandwidth can deal with cover traffic by using up as much of the allotted traffic between two nodes as possible with adversary-generated traffic [4]. As a result, no remaining bandwidth is available for legitimate cover traffic and the adversary can still deduce the amount of legitimate traffic that is being processed by the mix. This attack (as well as others described in this context) requires the adversary to have significant bandwidth. It should be noted that in contrast, the adversary described by our attack requires only the resources of an average mix operator.

Low-latency anonymity systems are also vulnerable to more active timing analysis variations. The attack presented in [23] is based on an adversary’s ability to track specific data through the network by making minor timing modifications to it. The attack assumes that the adversary controls the first and last nodes in the path through the network, with the goal of discovering which destination the initiator is communicating with. The authors discuss both correlating traffic “as is” as well as altering the traffic pattern at the first node in order to make correlation easier at the last node. For this second correlation attack, they describe a packet dropping technique which creates holes in the traffic; these holes then percolate through the network to the last router in the path. The analysis showed that without cover traffic (as employed in Tarzan [14, 15]) or defensive dropping [23], it is relatively easy to correlate communications through mix networks. Even with “normal” cover traffic where all packets between nodes look the same, Shmatikov and Wang show that the traffic analysis attacks are still viable [35]. Their proposed solution is to add cover traffic that mimics traffic flows from the initiator’s application.

A major limitation of all of the attacks described so far is that while they work well for small networks, they do not scale and may fail to produce reliable results for larger anonymizing networks. For example, Back’s active latency measuring attack [1] describes measuring the latencies of circuits and then trying to determine the nodes that were being utilized from the latency of a specific circuit. As the number of nodes grows, this attack

becomes more difficult (due to an increased number of possible circuits), especially as more and more circuits have similar latencies.

### 2.2.3 Congestion Attacks

A more powerful relative of the described timing attacks is the clogging or congestion attack. In a clogging attack, the adversary not only monitors the connection between two nodes but also creates paths through other nodes and tries to use all of their available capacity [1]; if one of the nodes in the target path is clogged by the attacker, the observed speed of the victim’s connection should change.

In 2005, Murdoch and Danezis described an attack on Tor [27] in which they could reveal all of the routers involved in a Tor circuit. They achieved this result using a combination of a circuit clogging attack and timing analysis. By measuring the load of each node in the network and then subsequently congesting nodes, they were able to discover which nodes were participating in a particular circuit. This result is significant, as it reduces Tor’s security during a successful attack to that of a collection of one hop proxies. This particular attack worked well on the fledgling Tor network with approximately fifty nodes; the authors experienced a high success rate and no false positives. However, their clogging attack no longer produces a signal that stands out on the current Tor network with thousands of nodes. Because today’s Tor network is more heavily used, circuits are created and destroyed more frequently, so the addition of a single clogging circuit has less impact. Also, the increased traffic transmitted through the routers leads to false positives or false negatives due to normal network fluctuations. We provide details about our attempt to reproduce Murdoch and Danezis’s work in Section 6.

McLachlan and Hopper [24] propose a similar circuit clogging attack against MorphMix [33], disproving claims made in [36] that MorphMix is invulnerable to such an attack. Because all MorphMix users are *required* to also be mix servers, McLachlan and Hopper achieve a stronger result than Murdoch and Danezis: they can identify not only the circuit, but the user as well.

Hopper et al. [19] build on the original clogging attack idea to construct a network latency attack to guess the location of Tor users. Their attack is two-phase: first use a congestion attack to identify the relays in the circuit, and then build a parallel circuit through those relays to estimate the latency between the victim and the first relay. A key contribution from their work is a more mathematical approach that quantifies the amount of information leaked in bits over time. We also note that without a working congestion attack, the practicality of their overall approach is limited.

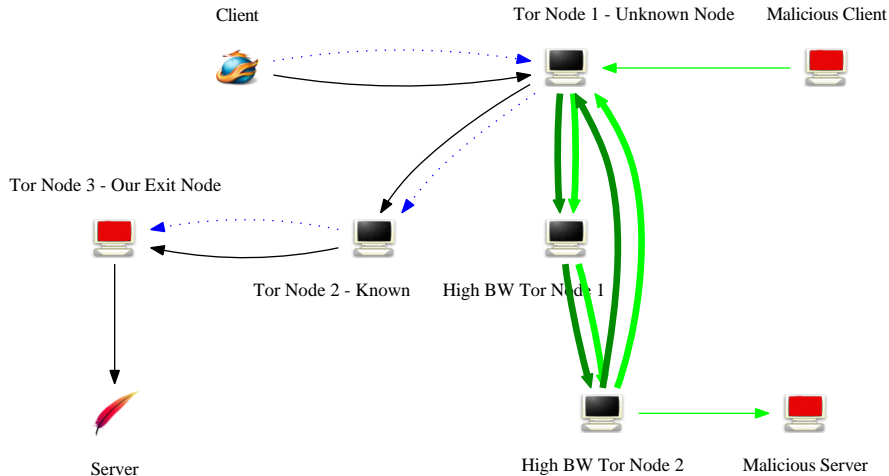


Figure 1: Attack setup. This figure illustrates the normal circuit constructed by the victim to the malicious Tor exit node and the “long” circuit constructed by the attacker to congest the entry (or guard) node used by the victim. The normal thin line from the *client* node to the *server* represents the victim circuit through the Tor network. The unwitting client has chosen the exit server controlled by the adversary, which allows the JavaScript injection. The double thick lines represent the long circular route created by the *malicious client* through the first Tor router chosen by the client. The dotted line shows the path that the JavaScript pings travel.

### 3 Our Attack

Three features of Tor’s design are crucial for our attack. First of all, Tor routers do not introduce any artificial delays when routing requests. As a result, it is easy for an adversary to observe changes in request latency. Second, the addresses of all Tor routers are publicly known and easily obtained from the directory servers. Tor developers are working on extensions to Tor (called bridge nodes [10, 11]) that would invalidate this assumption, but this service was not widely used at the time of this writing. Finally, the latest Tor server implementation that was available at the time we concluded our original attacks (Tor version 0.2.0.29-rc) did not restrict users from establishing paths of arbitrary length, meaning that there was no restriction in place to limit constructing long paths through Tor servers.<sup>1</sup> We used a modified client version (based on 0.2.0.22-rc) which used a small fixed path length (specifically three) but modified it to use a variable path length specified by our attacker.

Fig. 1 illustrates the three main steps of our attack. First, the adversary needs to ensure that the initiator repeatedly performs requests at known intervals. Second, the adversary observes the pattern in arrival times of these requests. Finally, the adversary changes the pattern by selectively performing a novel clogging attack on

<sup>1</sup>Tor version 0.2.1.3-alpha and later servers restrict path lengths to a maximum of eight because of this work.

Tor routers to determine the entry node. We will now describe each of these steps in more detail.

#### 3.1 JavaScript Injection

Our attack assumes that the adversary controls an exit node which is used by the victim to access an HTTP server. The attacker uses the Tor exit node to inject a small piece of JavaScript code (shown in Fig. 2) into an HTML response. It should be noted that most Tor users do **not** disable JavaScript and that the popular Tor Button plugin [31] and Privoxy [20] also do not disable JavaScript code; doing so would prevent Tor users from accessing too many web pages. The JavaScript code causes the browser to perform an HTTP request every second, and in response to each request, the adversary uses the exit node to return an empty response, which is thrown away by the browser. Since the JavaScript code may not be able to issue requests precisely every second, it also transmits the local system time (in milliseconds) as part of the request. This allows the adversary to determine the time difference between requests performed by the browser with sufficient precision. (Clock skew on the systems of the adversary and the victim is usually insignificant for the duration of the attack.) While JavaScript is not the only conceivable way for an attacker to cause a browser to transmit data at regular intervals (alternatives include HTTP headers like `refresh` [13]

```

<script language="javascript">
var count,timer,xmlhttp = 0;
function runonce() {
  xmlhttp = new XMLHttpRequest(); }
function start() {
  xmlhttp.abort();
  xmlhttp = new XMLHttpRequest();
  count++;
  if (timer) clearTimeout(timer);
  timer = setTimeout("start()", 1000);
  myDate = new Date();
  xmlhttp.open("GET",
    "/reportIn.html?num=" + count +
    "&time=" + myDate.getTime(),true);
  xmlhttp.send("");
}
</script>

```

Figure 2: JavaScript code injected by the adversary’s exit node. Note that other techniques such as HTML refresh, could also be used to cause the browser to perform periodic requests.

and HTML images [19]), JavaScript provides an easy and generally rather dependable method to generate such a signal.

The adversary then captures the arrival times of the periodic requests performed by the browser. Since the requests are small, an idle Tor network would result in the differences in arrival times being roughly the same as the departure time differences — these are known because they were added by the JavaScript as parameters to the requests. Our experiments suggest that this is often true for the real network, as most routers are not seriously congested most of the time. This is most likely in part due to TCP’s congestion control and Tor’s built-in load balancing features. Specifically, the variance in latency between the periodic HTTP requests without an active congestion attack is typically in the range of 0–5 s.

However, the current Tor network is usually not entirely idle and making the assumption that the victim’s circuit is idle is thus not acceptable. Observing congestion on a circuit is not enough to establish that the node under the congestion attack by the adversary is part of the circuit; the circuit may be congested for other reasons. Hence, the adversary needs to also establish a baseline for the congestion of the circuit without an active congestion attack. Establishing measurements for the baseline is done before and after causing congestion in order to ensure that observed changes during the attack are caused by the congestion attack and not due to unrelated changes in network characteristics.

The attacker can repeatedly perform interleaved mea-

surements of both the baseline congestion of the circuit and the congestion of the circuit while attacking a node presumed to be on the circuit. The attacker can continue the measurements until either the victim stops using the circuit or until the mathematical analysis yields a node with a substantially higher deviation from the baseline under congestion compared to all other nodes. Before we can describe details of the mathematical analysis, however, we have to discuss how congestion is expected to impact the latency measurements.

### 3.2 Impact of Congestion on Arrival Times

In order to understand how the congestion attack is expected to impact latency measurements, we first need to take a closer look at how Tor schedules data for routing. Tor makes routing decisions on the level of fixed-size *cells*, each containing 512 bytes of data. Each Tor node routes cells by going round-robin through the list of all circuits, transmitting one packet from each circuit with pending data (see Fig. 3). Usually the number of (active) circuits is small, resulting in little to no delay. If the number of busy circuits is large, messages may start to experience significant delays as the Tor router iterates over the list (see Fig. 4).

Since the HTTP requests transmitted by the injected JavaScript code are small (~250 bytes, depending on count and time), more than one request can fit into a single Tor cell. As a result multiple of these requests will be transmitted at the same time if there is congestion at a router. A possible improvement to our attack would be to use a lower level API to send the packets, as the XMLHttpRequest object inserts unnecessary headers into the request/response objects.

We will now characterize the network’s behavior under congestion with respect to request arrival times. Assuming that the browser transmits requests at a perfectly steady rate of one request per second, a congested router introducing a delay of (at most)  $n$  seconds would cause groups of  $n$  HTTP requests to arrive with delays of approximately  $0, 1, \dots, n-1$  seconds respectively: the first cell is delayed by  $n-1$  seconds, the cell arriving a second later by  $n-2$  seconds, and the  $n$ -th cell arrives just before the round-robin scheduler processes the circuit and sends all  $n$  requests in one batch. This characterization is of course a slight idealization in that it assumes that  $n$  is small enough to allow all of the HTTP requests to be grouped into one Tor cell and that there are no other significant fluctuations. Furthermore, it assumes that the amount of congestion caused by the attacker is perfectly steady for the duration of the time measurements, which may not be the case. However, even without these idealizations it is easy to see that the resulting latency histograms would still become “flat” (just not as perfectly

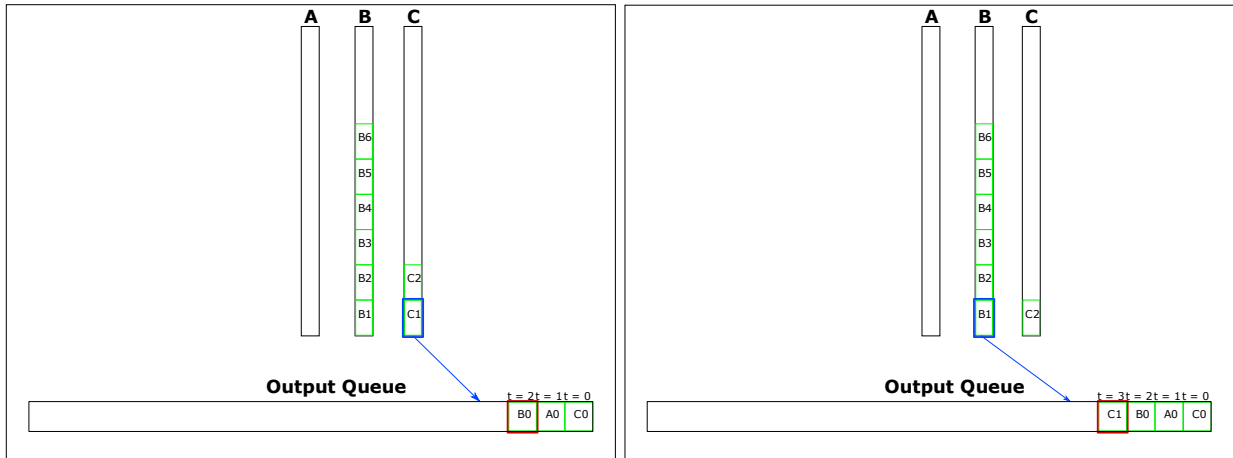


Figure 3: This example illustrates a Tor router which currently is handling three circuits at two points in time ( $t = 3$  and  $t = 4$ ). Circuits (A, B and C) have queues; cells are processed one at a time in a round-robin fashion. As the number of circuits increases, the time to iterate over the queues increases. The left figure shows the circuit queues and output queue before selection of cell C1 for output and the right figure shows the queues after queueing C1 for output. The thicker bottom box of queue C (left) and queue B (right) shows the current position of the round-robin queue iterator. At time  $t = 1$  the last cell from queue A was processed leaving the queue A empty. As a result, queue A is skipped after processing queue C.

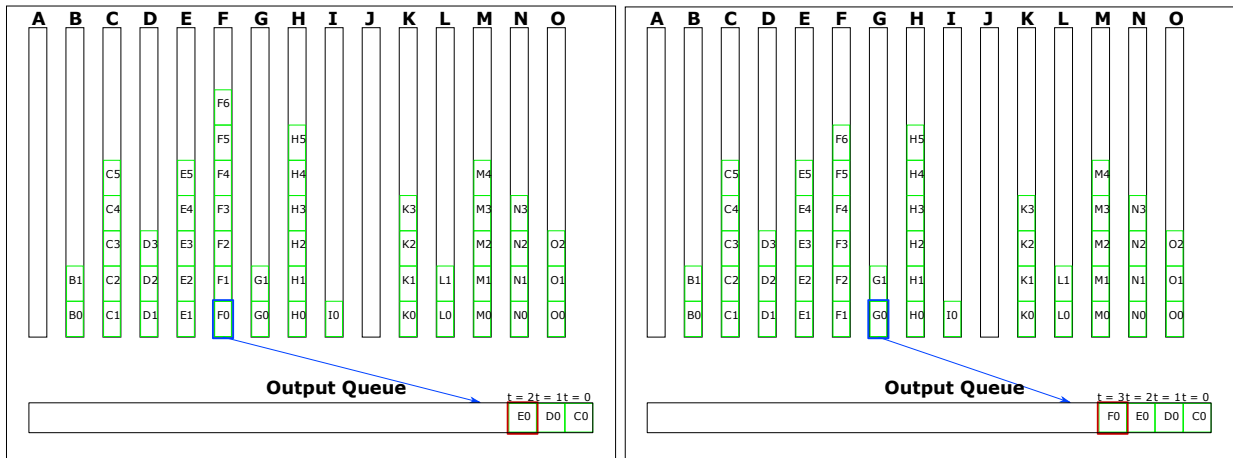


Figure 4: This example illustrates a Tor router under congestion attack handling 15 circuit queues. Note that if a circuit includes a node multiple times, the node assigns the circuit multiple circuit queues. In this example, not all of the circuit queues are busy — this may be because the circuits are not in use or because other routers on the circuit are congested. As in Fig. 3, the left and right figures show the state of the mix before and after queueing a cell, in this case F0.

regular in terms of arrival patterns) assuming the load caused by the attacker is sufficiently high.

Since we ideally expect delays in message arrival times for a congested circuit to follow a roughly flat distribution between zero and  $n$ , it makes sense to compute a histogram of the delays in message arrival times. If the congestion attack is targeting a node on the circuit, we would expect to see a roughly equal number of messages in each interval of the histogram. We will call the shape of the resulting histogram *horizontal*. If the circuit is not congested, we expect to see most messages arrive without significant delay which would place them in the bucket for the lowest latency. We will call the shape of the resulting histogram *vertical*. So for example, in Fig. 6 the control data are vertical, whereas the attack data are more horizontal.

Note that the clock difference between the victim’s system and the adversary as well as the minimal network delay are easily eliminated by normalizing the observed time differences. As a result, the latency histograms should use the increases in latency over the smallest observed latency, not absolute latencies.

### 3.3 Statistical Evaluation

In order to numerically capture congestion at nodes we first measure the node’s *baseline* latency, that is latency without an active congestion attack (at least as far as we know). We then use the observed latencies to create  $n$  bins of latency intervals such that each bin contains the same number of data points. Using the  $\chi^2$ -test we could then determine if the latency pattern at the respective peer has changed “significantly”. However, this simplistic test is insufficient. Due to the high level of normal user activity, nodes frequently do change their behavior in terms of latencies, either by becoming congested or by congestion easing due to clients switching to other circuits. For the attacker, congestion easing (the latency histogram getting more vertical) is exactly the opposite of the desired effect. Hence the ordinary  $\chi^2$  test should not be applied without modification. What the attacker is looking for is the histogram becoming more horizontal, which for the distribution of the bins means that there are fewer values in the low-latency bins and more values in the high-latency bins. For the medium-latency bins no significant change is expected (and any change there is most likely noise).

Hence we modify our computation of the  $\chi^2$  value such that we only include changes in the anticipated direction: for the bins corresponding to the lowest third of the latencies, the square of the difference between expected and observed number of events is only counted in the summation if the number of observed events is lower than expected. For the bins corresponding to the high-

est third of the latencies, the square of the difference between expected and observed number of events is only counted if the number of observed events is higher than expected. Since changes to the bins in the middle third are most likely noise, those bins are not included in the  $\chi^2$  calculation at all (except as a single additional degree of freedom).

Using this method, a single iteration of measuring the baseline and then determining that there was a significant increase in latency (evidenced by a large  $\chi^2$ -value), only signifies that congestion at the guard for the victim circuit was correlated (in time) with the congestion caused by the attacker. Naturally, correlation does not imply causality; in fact, for short (30–60 s) attack runs it frequently happens that the observed  $\chi^2$ -value is higher for some false-positive node than when attacking the correct guard node. However, such accidental correlations virtually never survive iterated measurements of the latency baseline and  $\chi^2$ -values under attack.

### 3.4 Congestion Attack

Now we focus on how the attacker controlling the exit node of the circuit will cause significant congestion at nodes that are suspected to be part of the circuit. In general, we will assume that all Tor routers are suspects and that in the simplest case, the attacker will iterate over all known Tor routers with the goal of finding which of these routers is the entry point of the circuit.

For each router  $X$ , the attacker constructs a long circuit that repeatedly includes  $X$  on the path. Since Tor relays will tear down a circuit that tries to extend to the previous node, we have to use two (or more) other (preferably high-bandwidth) Tor routers before looping back to  $X$ . Note that the attacker could choose two different (involuntary) helper nodes in each loop involving  $X$ . Since  $X$  does not know that the circuit has looped back to  $X$ , Tor will treat the long attack circuit as many different circuits when it comes to packet scheduling (Fig. 4).

Once the circuit is sufficiently long (we typically found 24 hops to be effective, but in general this depends on the amount of congestion established during the baseline measurements), the attacker uses the circuit to transmit data. Note that a circuit of length  $m$  would allow an attacker with  $p$  bandwidth to consume  $m \cdot p$  bandwidth on the Tor network, with  $X$  routing as much as  $\frac{m \cdot p}{3}$  bandwidth. Since  $X$  now has to iterate over an additional  $\frac{m}{3}$  circuits, this allows the attacker to introduce large delays at this specific router. The main limitation for the attacker here is time. The larger the desired delay  $d$  and the smaller the available attacker bandwidth  $p$  the longer it will take to construct an attack circuit of sufficient length  $m$ : the number of times that the victim node is part of the attack circuit is proportional to the length of

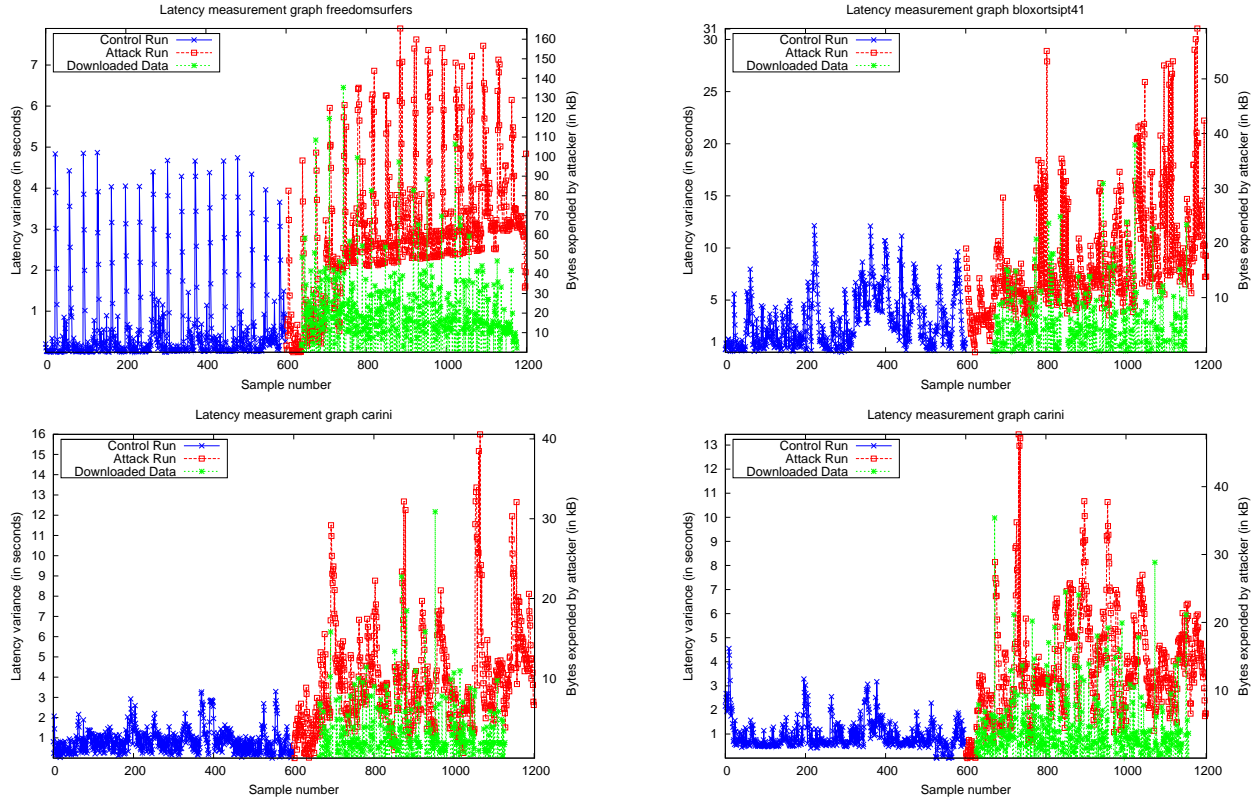


Figure 5: These figures show the results of perturbation of circuits in Tor and the resulting effects on latency. The  $x$ -axes show sample numbers (one per second), and the (left)  $y$ -axes are latency variance observed on the circuits in seconds. The attack on the first router of each circuit starts at time 600; the third line shows the amount of data (scaled) that transferred through the attack circuit (scaled to the right  $y$ -axes). These are individual trials; each shows a single control run and a single attack run.

the circuit  $m$ . In other words, the relationship between  $p$ ,  $m$  and the delay  $d$  is  $d \sim p \cdot m$ .

If the router  $X$  is independent of the victim circuit, the measured delays should not change significantly when the attack is running. If  $X$  is the entry node, the attacker should observe a delay pattern that matches the power of the attack – resulting in a horizontal latency variance histogram as described in Section 3.2. The attacker can vary the strength of the attack (or just switch the long attack circuit between idle and busy a few times) to confirm that the victim’s circuit latency changes correlate with the attack. It should be noted that the attacker should be careful to not make the congestion attack too powerful, especially for low-bandwidth targets. In our experiments we sometimes knocked out routers (for a while) by giving them far too much traffic. As a result, instead of receiving requests from the JavaScript code with increasing latencies, the attacker suddenly no longer receives requests at all, which gives no useful data for the statistical evaluation.

### 3.5 Optimizations

The adversary can establish many long circuits to be used for attacks before trying to deanonymize a particular victim. Since idle circuits would not have any impact on measuring the baseline (or the impact of using another attack circuit), this technique allows an adversary to eliminate the time needed to establish circuits. As users can only be expected to run their browser for a few minutes, eliminating this delay may be important in practice; even users that may use their browser for hours are likely to change between pages (which might cause Tor to change exit nodes) or disable Tor.

In order to further speed up the process, an adversary can try to perform a binary search for  $X$  by initially running attacks on half of the routers in the Tor network. With pre-built attack circuits adding an almost unbounded multiplier to the adversary’s resources, it is conceivable that a sophisticated attacker could probe a network of size  $s$  in  $\log_2 s$  rounds of attacks.

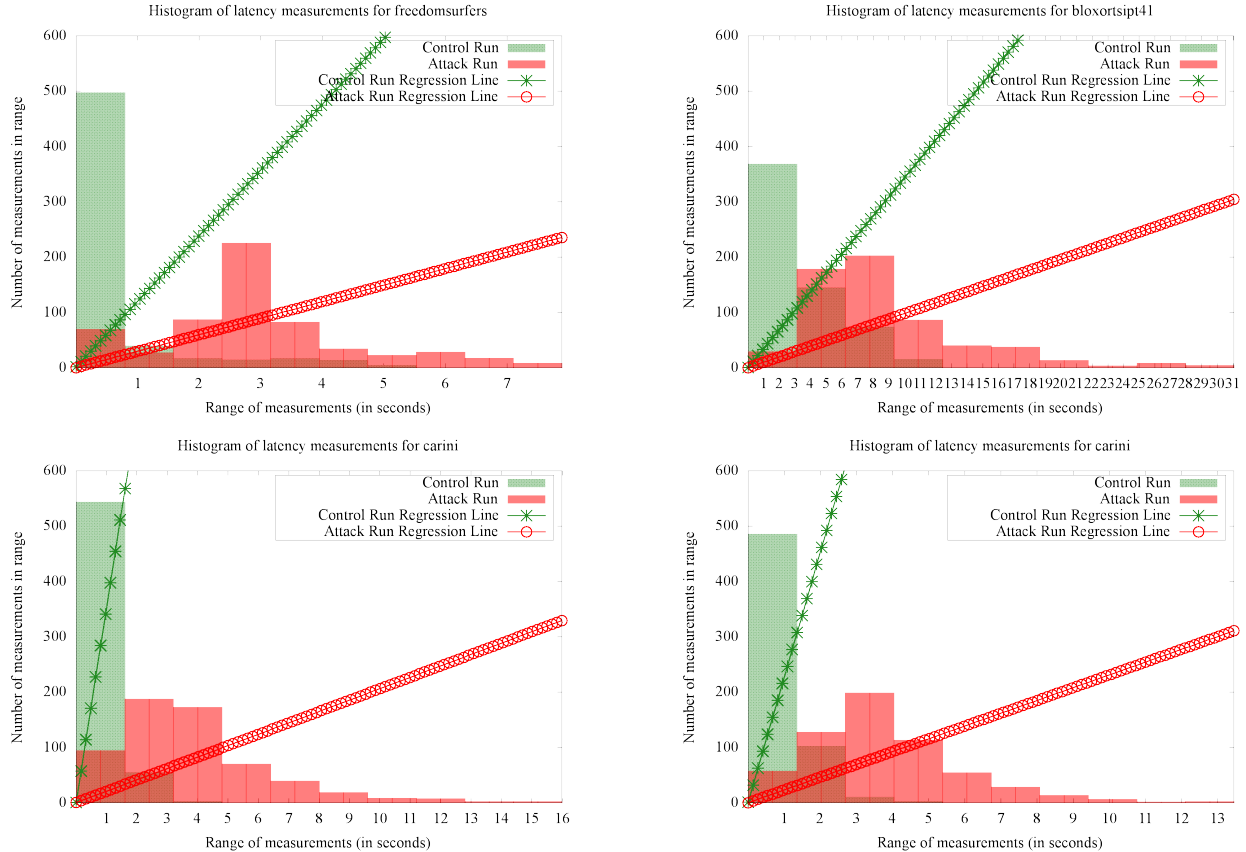


Figure 6: These figures show the results of four independent runs of our congestion attack. In the histograms the  $x$ -axis groups ranges of latency variance values together and the  $y$ -axis represents the number of readings received in that range. The hash marked bars represent the unperturbed measurements on a circuit and the plain bars show measurements from the same circuit during the attack. The effect of the attack is a shift to higher latency values. The first and second lines are linear least squares fit approximations for the baseline and congestion runs, respectively. These data show the difference between a single control/attack run and are not averages of many runs.

In practice, pre-building a single circuit that would cause congestion for half the network is not feasible; the Tor network is not stable enough to sustain circuits that are thousands of hops long. Furthermore, the differences in available bandwidth between the routers complicates the path selection process. In practice, an adversary would most likely pre-build many circuits of moderate size, forgoing some theoretical bandwidth and attack duration reductions for circuits that are more reliable. Furthermore, the adversary may be able to exclude certain Tor routers from the set of candidates for the first hop based on the overall round-trip latency of the victim’s circuit. The Tor network allows the adversary to measure the latency between any two Tor routers [19, 27]; if the overall latency of the victim’s circuit is smaller than the latency between the known second router on the path and another router  $Y$ , then  $Y$  is most likely not a candi-

date for the entry point.

Finally, the adversary needs to take into consideration that by default, a Tor user switches circuits every 10 minutes. This further limits the window of opportunity for the attacker. However, depending on the browser, the adversary may be able to cause the browser to pipeline HTTP requests which would not allow Tor to switch circuits (since the HTTP session would not end). Tor’s circuit switching also has advantages for the adversary: every 10 minutes there is a new chance that the adversary-controlled exit node is chosen by a particular victim. Since users only use a small number of nodes for the first node on a circuit (these nodes are called guard nodes [29]), the adversary has a reasonable chance over time to determine these guard nodes. Compromising one of the guard nodes would then allow full deanonymization of the target user.

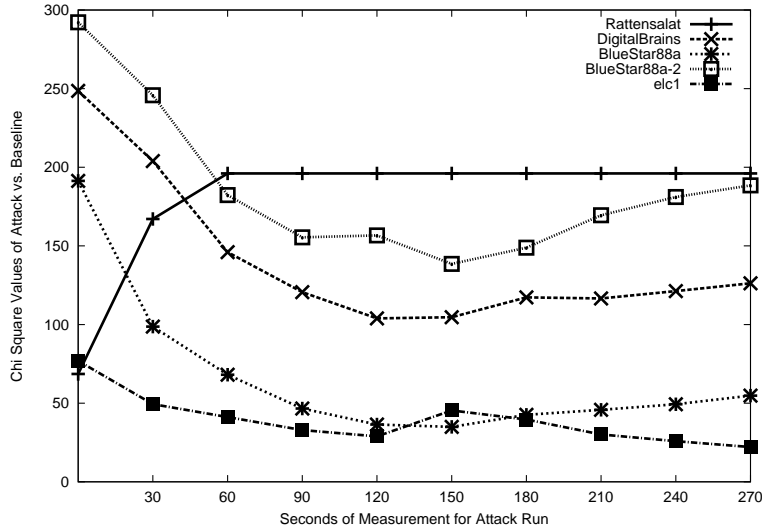


Figure 7: This figure shows the development of  $\chi^2$  values (using the modified  $\chi^2$  calculation as described in Section 3.3) for the various candidates over a prolonged period of performing a congestion attack on the various nodes. The  $\chi^2$  values are computed against a five-minute baseline obtained just prior to the congestion attack. The  $\chi^2$  value of the correct entry node quickly rises to the top whereas the  $\chi^2$  values for all of the other candidates are typically lower after about a minute of gathering latency information under congestion. This illustrates that a few minutes are typically sufficient to obtain a meaningful  $\chi^2$  value.

## 4 Experimental Results

The results for this paper were obtained by attacking Tor routers on the real, deployed Tor network (initial measurements were done during the Spring and Summer of 2008; additional data was gathered in Spring 2009 with an insignificantly modified attacker setup; the modifications were needed because our original attack client was too outdated to work with the majority of Tor routers at the time). In order to confirm the accuracy of our experiments and avoid ethical problems, we did not attempt to deanonymize real users. Instead, we established our own client circuits through the Tor network to our malicious exit node and then confirmed that our statistical analysis was able to determine the entry node used by our own client. Both the entry nodes and the second nodes on the circuits were normal nodes in the Tor network outside of our control.

The various roles associated with the adversary (exit node, malicious circuit client, and malicious circuit webserver) as well as the “deanonymized” victim were distributed across different machines in order to minimize interference between the attacking systems and the targeted systems. For the measurements we had the simulated victim running a browser requesting and executing the malicious JavaScript code, as well as a machine running the listening server to which the client transmits the “ping” signal approximately every second (Fig. 1). The

browser always connected to the same unmodified Tor client via Privoxy [20]. The Tor client used the standard configuration except that we configured it to use our malicious exit node for its circuits. The other two nodes in the circuit were chosen at random by Tor. Our malicious exit node participated as a normal Tor router in the Tor network for the duration of the study (approximately six weeks). For our tests we did not actually make the exit server inject the JavaScript code; while this is a relatively trivial modification to the Tor code we used a simplified setup with a webserver serving pages with the JavaScript code already present.

The congestion part of the attack requires three components: a simple HTTP server serving an “infinite” stream of random data, a simple HTTP client downloading this stream of data via Tor, and finally a modified Tor client that constructs “long” circuits through those Tor nodes that the attacker would like to congest. Specifically, the modified Tor client allows the attacker to choose two (or more) routers with high bandwidth and a specific target Tor node, and build a long circuit by repeatedly alternating between the target node and the other high bandwidth nodes. The circuit is eventually terminated by connecting from some high-bandwidth exit node to the attacker’s HTTP server which serves the “infinite” stream of random data as fast as the network can process it. As a result, the attacker maximizes the utilization of the Tor circuit. Naturally, an attacker with

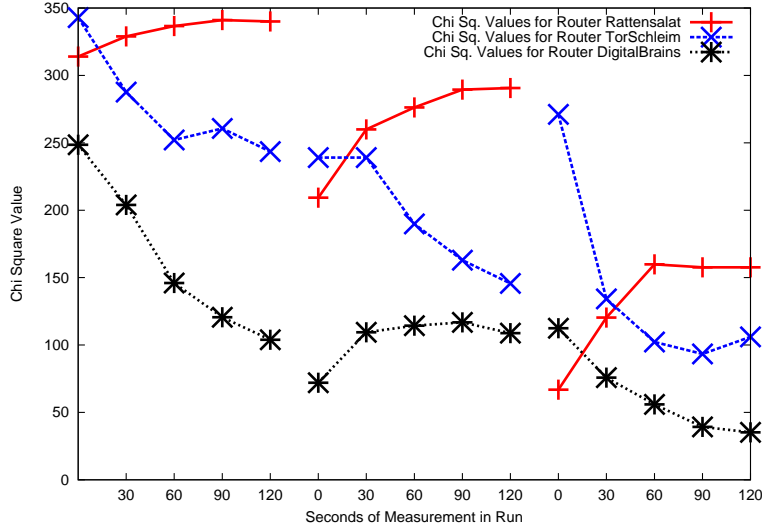


Figure 8: This graph shows three sets of cumulative  $\chi^2$  computations for three nodes; the actual entry node (Rattensalat), a node that initially shows up as a false-positive (TorSchleim) and a typical negative (DigitalBrains). As expected, the  $\chi^2$  values (at time 120 s) are consistently the highest for the correct node; false-positives can be ruled out through repeated measurements.

significant bandwidth can elect to build multiple circuits in parallel or build shorter circuits and still exhaust the bandwidth resources of the target Tor router.

In order to cause congestion, we simply started the malicious client Tor process with the three chosen Tor routers and route length as parameters and then attempted to connect via `libcurl` [6] to the respective malicious server process. The amount of data received was recorded in order to determine bandwidth consumed during the tests. In order to further increase the load on the Tor network the experiments presented actually used two identical attacker setups with a total of six machines duplicating the three machine setup described in the previous paragraph. We found path lengths of 24 (making our attack strength eight times the attacker bandwidth) sufficient to alter latencies. The overall strength of the attack was measured by the sum of the number of bytes routed through the Tor network by both attacker setups. For each trial, we waited to receive six hundred responses from the “victim”; since the browser transmitted requests to Tor at roughly one request per second, a trial typically took approximately ten minutes.

In addition to measuring the variance in packet arrival time while congesting a particular Tor router, each trial also included baseline measurements of the “uncongested” network to discover the normal variance in packet arrival time for a particular circuit. As discussed earlier, these baseline measurements are crucial for determining the significance of the effect that the congestion attack has had on the target circuit.

Fig. 5 illustrates how running the attack on the first hop of a circuit changes the latency of the received HTTP requests generated by the JavaScript code. The figure uses the same style chosen by Murdoch and Danezis [27], except that an additional line was added to indicate the strength of the attack (as measured by the amount of traffic provided by the adversary). For comparison, the first half of each of the figures shows the node latency variance when it is *not* under active congestion attack (or at least not by us).

While the plots in Fig. 5 visualize the impact of the congestion attack in a simple manner, histograms showing the variance in latency are more suitable to demonstrate the significance of the statistical difference in the traffic patterns. Fig. 6 shows the artificial delay experienced by requests traveling through the Tor network as observed by the adversary. Since Tor is a low-latency anonymization service, the requests group around a low value for a circuit that is not under attack. As expected, if the entry node is under attack, the delay distribution changes from a steep vertical peak to a mostly horizontal distribution. Fig. 6 also includes the best-fit linear approximation functions for the latency histograms which we use to characterize how vertical or how horizontal the histogram is as described in Section 3.2.

Fig. 7 illustrates how the  $\chi^2$  values evolve for various nodes over time. Here, we first characterized the baseline congestion for the router for five minutes. Then, the congestion attack was initiated (congesting the various guard nodes). For each attacked node, we used the modified

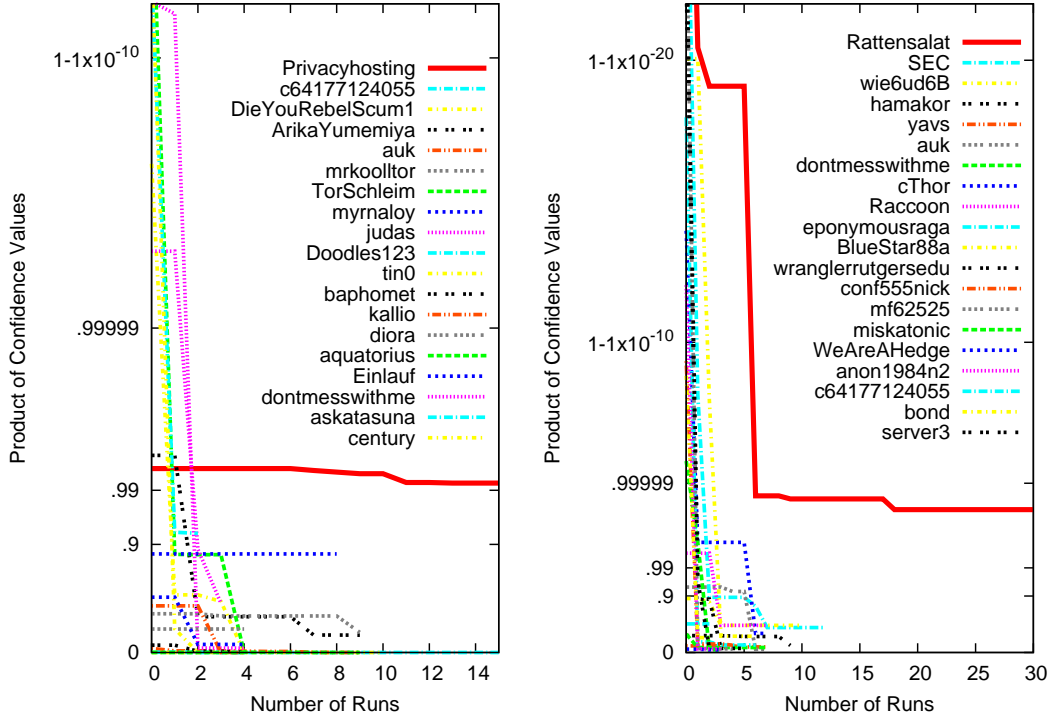


Figure 9: Plot of the product of  $\chi^2$   $p$ -values for the top 20 candidate nodes (out of  $\sim 200$  and  $\sim 250$ , respectively) by run (a run is 300 s baseline vs. 300 s attack) for two entry nodes. The first hop (*Privacyhosting* (left), *Rattensalat* (right)) routers were tested many more times than other routers, because the others quickly fell to low values. We expect an attacker to perform more measurements for routers that score high to validate the correct entry node was found. Our measurements demonstrate that the multiplied  $p$ -value remains consistently high for the correct entry node. The  $y$ -axis is plotted on a log scale from 0 to  $1 - 1 \times 10^{-10}$  and  $1 - 1 \times 10^{-20}$ , respectively. We speculate that the lower maximum value for *Privacyhosting* is due to its higher bandwidth (900 kB/s vs. 231 kB/s).

$\chi^2$  summation (from Section 3.3) to determine how congested the victim’s circuit had become at that time. We computed (cumulative)  $\chi^2$  values after 30 s, 60 s, 90 s and so forth. For the  $\chi^2$  calculations, we used 60 bins for 300 baseline values; in other words, the time intervals for the bins were chosen so that each bin contained five data points during the five minutes of baseline measurement. The 20 bins in the middle were not included in the summation, resulting in 40 degrees of freedom. As expected, given only 30 s of attack data some “innocent” nodes have higher  $\chi^2$  values compared to the entry node (false-positives). However, given more samples the  $\chi^2$  values for those nodes typically drop sharply whereas the  $\chi^2$  value when congesting the entry node increases or remains high. Of course, false-positive nodes  $\chi^2$  values may increase due to network fluctuations over time as well.

Unlucky baseline measurements and shifts in the baseline latency of a router over time can be addressed by iterating between measuring baseline congestion and at-

tack measurements. Fig. 8 shows three iterations of first determining the current baseline and then computing  $\chi^2$  values under attack. Again the correct entry node exhibits the largest  $\chi^2$  values each time after about a minute of gathering latency data under attack.

Given the possibility of false-positives showing up initially when computing  $\chi^2$  values, the attacker should target “all” suspected guard nodes for the first few iterations, and then focus his efforts on those nodes that scored highly. Fig. 9 illustrates this approach. It combines the data from multiple iterations of baseline measurements and  $\chi^2$  calculations from attack runs. The attacker determines for each  $\chi^2$  value the corresponding confidence interval. These values are frequently large (99.9999% or higher are not uncommon) since Tor routers do frequently experience significant changes in congestion. Given these individual confidence values for each individual iteration, a cumulative score is computed as the product<sup>2</sup> of these values. Fig. 9 shows the Tor

<sup>2</sup>It is conceivable that multiplying  $\chi^2$  values may cause false-

Router	$\Pi p$	$r$	Peak BW	Configured BW
Rattensalat	0.999991	44	231 kB/s	210 kB/s
c64177124055	0.903	3	569 kB/s	512 kB/s
Raccoon	0.891	8	3337 kB/s	4100 kB/s
wie6ud6B	0.890	11	120 kB/s	100 kB/s
SEC	0.870	13	4707 kB/s	5120 kB/s
cThor	0.789	8	553 kB/s	500 kB/s
BlueStar88a	0.734	7	111 kB/s	100 kB/s
bond	0.697	3	407 kB/s	384 kB/s
eponymousraga	0.458	7	118 kB/s	100 kB/s
conf555nick	0.450	5	275 kB/s	200 kB/s

Table 1: This table lists the top ten (out of 251 total) products of confidence intervals ( $p$ -values).  $r$  is the number of iterations (and hence the number of factors in  $\Pi p$ ) that was performed for the respective router. As expected, the entry node Rattensalat achieves the highest score.

routers with the highest cumulative scores using this metric from trials on two different entry nodes. Note that fewer iterations were performed for routers with low cumulative scores; the router with the highest score (after roughly five iterations) and the most overall iterations is the correctly identified entry node of the circuit.

Table 1 contrasts the product of  $\chi^2$  values (as introduced in Section 3.3) obtained while attacking the actual first hop with the product while attacking other Tor routers. The data shows that our attack can be used to distinguish the first hop from other routers when controlling the exit router (therefore knowing a priori the middle router).

Finally, by comparing the highest latency observed during the baseline measurement with the highest latency observed under attack, Table 2 provides a simple illustration showing that the congestion attack actually has a significant effect.

## 5 Proposed Solutions

An immediate workaround that would address the presented attack would be disabling of JavaScript by the end user. However, JavaScript is not the only means by which an attacker could obtain timing information. For example, redirects embedded in the HTML header could also be used (they would, however, be more visible to the end user). Links to images, frames and other features of HTML could also conceivably be used to generate repeated requests. Disabling all of these features has the disadvantage that the end user’s browsing experience would suffer.

---

negatives should a single near-zero  $\chi^2$  value for the correct entry node be observed. While we have not encountered this problem in practice, using the mean of  $\chi^2$  values would provide a way to avoid this theoretical problem.

A better solution would be to thwart the denial-of-service attack inherent in the Tor protocol. Attackers with limited bandwidth would then no longer be able to significantly impact Tor’s performance. Without the ability to selectively increase the latency of a particular Tor router, the resulting timing measurements would most likely give too many false positives. We have extended the Tor protocol to limit the length of a path. The details are described in [9]; we will detail the key points here.

In the modified design, Tor routers now must keep track of how often each circuit has been extended and refuse to route messages that would extend the circuit beyond a given threshold  $t$ . This can be done by tagging messages that *may* extend the circuit with a special flag that is not part of the encrypted stream. The easiest way to do this is to introduce a new Tor cell type that is used to flag cells that may extend the circuit. Routers then count the number of messages with the special flag and refuse to route more than a given small number (at the moment, eight) of those messages. Routers that receive a circuit-extension request check that the circuit-extension message is contained in a cell of the appropriate type. Note that these additional checks do not change the performance characteristics of the Tor network. An attacker could still create a long circuit by looping back to an adversary-controlled node every  $t$  hops; however, the adversary would then have to provide bandwidth to route every  $t$ -th packet; as a result, the bandwidth consumption by the attacker is still bounded by the small constant  $t$  instead of the theoretically unbounded path length  $m$ .

While this change prevents an attacker from constructing a circuit of arbitrary length, it does not fully prevent the attacker from constructing a path of arbitrary length. The remaining problem is that the attacker could establish a circuit and then from the exit node reconnect to the Tor network again as a client. We could imagine config-

Router Attacked	Max Latency Difference	Avg. Latency Difference	Runs
Rattensalat	70352 ms	25822 ms	41
Wiia	46215 ms	470 ms	5
downtownzion	39522 ms	2625 ms	9
dontmesswithme	37648 ms	166 ms	8
wie6ud6B	35058 ms	9628 ms	9
TorSchleim	28630 ms	5765 ms	15
hamakor	25975 ms	6532 ms	8
Vault24	24330 ms	4647 ms	7
Einlauf	22626 ms	2017 ms	8
grsrlfz	22545 ms	10112 ms	2

Table 2: This table shows the top 10 highest latency differences between the maximum observed measurement in attack runs versus the baseline runs for each router. Unsurprisingly, the difference between the maximum latency observed during the congestion attack and the baseline measurement is significantly higher when attacking the correct first hop compared to attacking other routers. Also included for comparison is the average max latency over all iterations (also higher for the correct first hop), and the number of runs.

uring all Tor relays to refuse incoming connections from known exit relays, but even this approach does not entirely solve the problem: the attacker can use any external proxies he likes (e.g. open proxies, unlisted Tor relays, other anonymity networks) to “glue” his circuits together. Assuming external proxies with sufficient aggregate bandwidth are available for gluing, he can build a chain of circuits with arbitrary length. Note that the solution proposed in [30] — limiting circuit construction to trees — does not address this issue; furthermore, it increases overheads and implementation complexity far beyond the change proposed here and (contrary to the claims in [30]) may also have an impact on anonymity, since it requires Tor to fundamentally change the way circuits are constructed. We leave a full solution to this problem as an open research question.

Finally, given that strong adversaries may be able to mount latency altering attacks without Tor’s “help”, Tor users might consider using a longer path length than the minimalistic default of three. This would involve changes to Tor, as currently the only way for a user to change the default path length would be to edit and recompile the code (probably out of scope for a “normal” user). While the presented attack can be made to work for longer paths, the number of false positives and the time required for a successful path discovery increase significantly with each extra hop. Using a random path length between four and six would furthermore require the adversary to confirm that the first hop was actually found (by determining that none of the other Tor routers could be a predecessor). Naturally, increasing the path length from three to six would significantly increase the latency and bandwidth requirements of the Tor network and might also hurt with respect to other attacks [2].

## 6 Low-cost Traffic Analysis Failure Against Modern Tor

We attempted to reproduce Murdoch and Danezis’s work [27] on the Tor network of 2008. Murdoch provided us with their code and statistical analysis framework which performs their congestion attack while measuring the latency of the circuit. Their analysis also determines the average latency and uses normalized latencies as the strength of the signal.

The main difference in terms of how data is obtained between Murdoch and Danezis and the attack presented in Section 3 is that Murdoch and Danezis use a circuit constructed by the attacker to measure the latency introduced by the victim circuit whereas our attack uses a circuit constructed by the victim to measure the latency introduced by the attacker.

As in this paper, the adversary implemented by Murdoch and Danezis repeatedly switches the congestion attack on and off; a high correlation between the presence of high latency values and the congestion attack being active is used to determine that a particular router is on the circuit. If such a correlation is absent for the correct router, the attack produces false negatives and fails. If a strong correlation is present between high latency values and random time periods (without an active attack) then the attack produces false positives and also fails.

Fig. 10 shows examples of our attempts at the method used in [27], two with the congestion attack being active and two without. Our experiments reproduced Murdoch and Danezis’s attack setup where the attacker tries to measure the congestion caused by the victim’s circuit. Note that in the graphs on the right, the congestion attack was run against a Tor router unrelated to the circuit

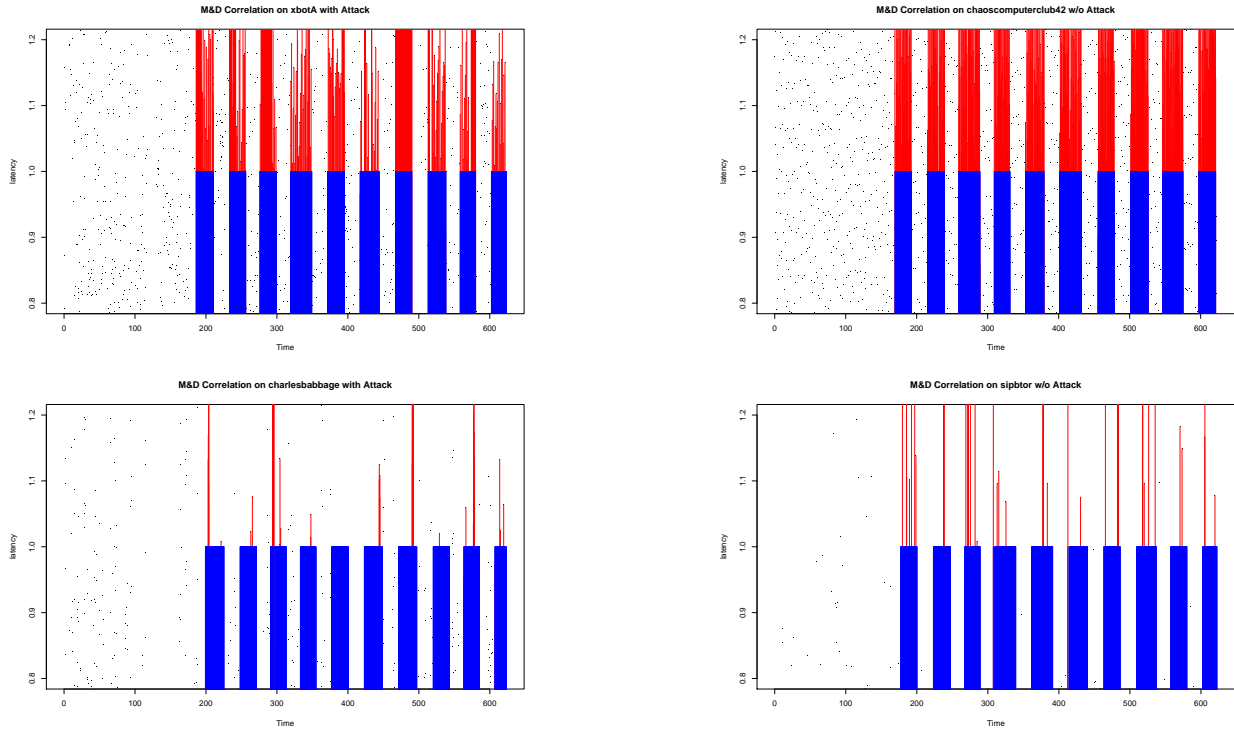


Figure 10: These graphs show four runs of the method used in [27], two with the congestion attack being active (on the left) and two without (on the right). The figure plots the observed latency of a router over time. Blue bars are used to indicate when the congestion attack was active; in the case of the graphs on the right the attack was active on an unrelated circuit. Red lines are drawn to latency values above average to mark latencies that correlate with the attack, according to the Murdoch and Danezis style analysis.

and thus inactive for the circuit that was measured. Any correlation observed in this case implies that Murdoch and Danezis’s attack produces false positives. The “visual” look of the graphs is the same whether the attack is targeted at that relay or not. Specifically, the graphs on the right suggest a similar correlation pattern even when the attack was “off” (or targeting unrelated Tor routers). This is due to the high volume of traffic on today’s Tor network causing baseline congestion which makes their analysis too indiscriminate.

Table 3 shows some representative correlation values that were computed using the statistical analysis from [27] when performed on the modern Tor network. Note that the correlation values are high regardless of whether or not the congestion attack was actually performed on the respective router. For Murdoch and Danezis’s analysis to work, high correlation values should only appear for the attacked router.

The problem with Murdoch and Danezis’s attack and analysis is not primarily with the statistical method; the single-circuit attack itself is simply not generating a sufficiently strong signal on the modern network. Fig. 11

plots the baseline latencies of Tor routers as well as the latencies of routers subjected to Murdoch and Danezis’s congestion attack in the style we used in Fig. 6. There are hardly any noticeable differences between routers under Murdoch and Danezis’s congestion attack and the baseline. Fig. 12 shows the latency histograms for the same data; in contrast to the histograms in Fig. 6 there is little difference between the histograms for the baseline and the attack.

In conclusion, due to the large amount of traffic on the modern Tor network, Murdoch and Danezis’s analysis is unable to differentiate between normal congestion and congestion caused by the attacker; the small amount of congestion caused by Murdoch and Danezis is lost in the noise of the network. As a result, their analysis produces many false positives and false negatives. While these experiments only represent a limited case-study and while Murdoch and Danezis’s analysis may still work in some cases, we never got reasonable results on the modern Tor network.

Router	Correlation	Attacked?	Peak BW	Configured BW
morphiumpherrex	1.43	Yes	222 kB/s	201 kB/s
chaoscomputerclub23	1.34	No	5414 kB/s	5120 kB/s
humanistischeunion1	1.18	No	5195 kB/s	6000 kB/s
mikezhangwithtor	1.07	No	1848 kB/s	2000 kB/s
hummingbird	1.03	No	710 kB/s	600 kB/s
chaoscomputerclub42	1.00	Yes	1704 kB/s	5120 kB/s
degaussYourself	1.00	No	4013 kB/s	4096 kB/s
ephemera	0.91	Yes	445 kB/s	150 kB/s
fissefjaes	0.99	Yes	382 kB/s	50 kB/s
zymurgy	0.86	Yes	230 kB/s	100 kB/s
charlesbabbage	0.53	Yes	2604 kB/s	1300 kB/s

Table 3: This table shows the correlation values calculated using the Murdoch and Danezis’s attack on the Tor network in Spring of 2008. False positives and false negatives are both abundant; many naturally congested routers show a strong correlation suggesting they are part of the circuit when they are not.

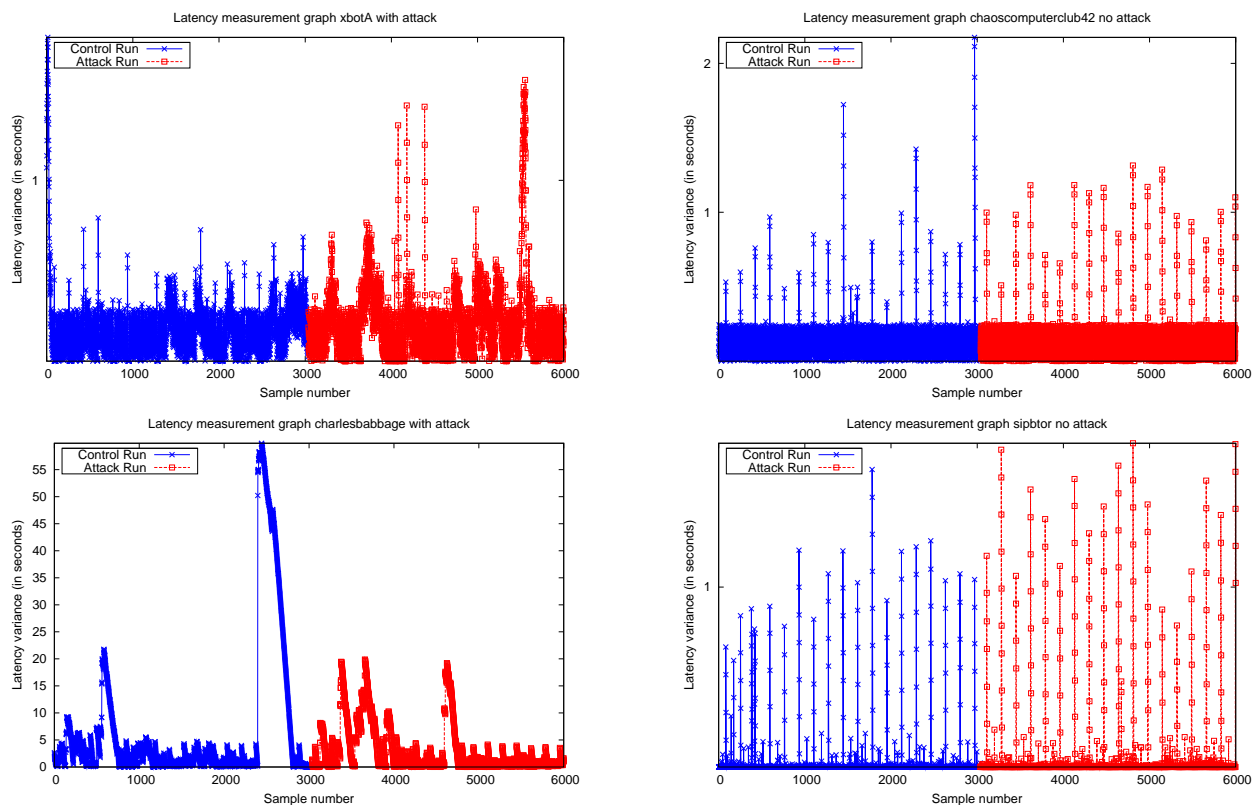


Figure 11: These graphs correspond to Fig. 10, showing the same attack in the style we used in Fig. 5. Note that during the attack phase the congestion circuit is turned on and off just as illustrated in Fig. 10. For all four routers the latency measurements are almost identical whether the attack was present or not.

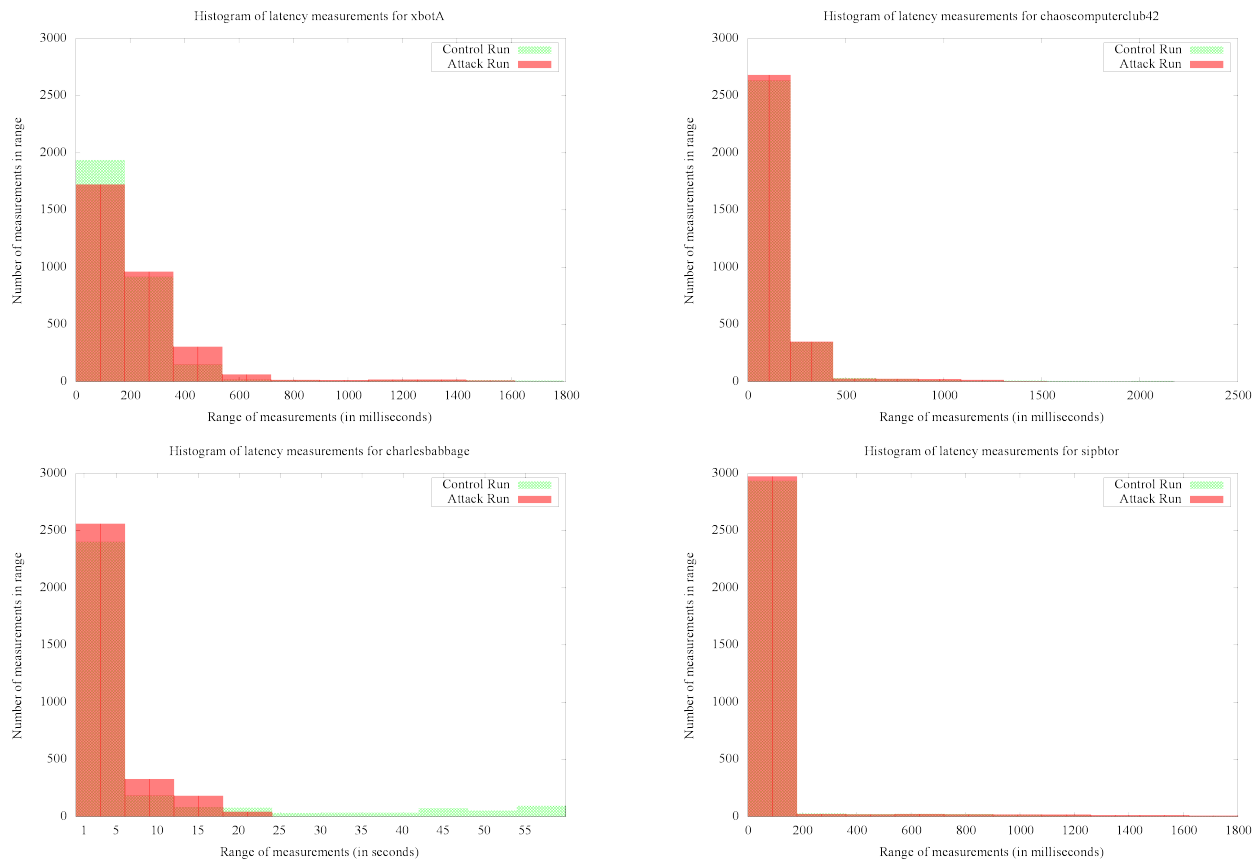


Figure 12: Once more we show the same data for comparison as shown in Fig. 10, this time in the histogram style we use in Fig. 6. The overlap between the control run and the attack run is difficult to see due to the similarity of latency distributions.

## 7 Conclusion

The possibility of constructing circuits of arbitrary length was previously seen as a minor problem that could lead to a DoS attack on Tor. This work shows that the problem is more serious, in that an adversary could use such circuits to improve methods for determining the path that packets take through the Tor network. Furthermore, Tor’s minimalistic default choice to use circuits of length three is questionable, given that an adversary controlling an exit node would only need to recover a tiny amount of information to learn the entire circuit. We have made some minimal changes to the Tor protocol that make it more difficult (but not impossible) for an adversary to construct long circuits.

## Acknowledgments

This research was supported in part by the NLnet Foundation from the Netherlands (<http://nlnet.nl/>)

and under NSF Grant No. 0416969. The authors thank P. Eckersley for finding a problem in an earlier draft of the paper and K. Grothoff for editing.

## References

- [1] BACK, A., MÖLLER, U., AND STIGLIC, A. Traffic analysis attacks and trade-offs in anonymity providing systems. In *Proceedings of Information Hiding Workshop (IH 2001)* (April 2001), I. S. Moskowitz, Ed., Springer-Verlag, LNCS 2137, pp. 245–257.
- [2] BORISOV, N., DANEZIS, G., MITTAL, P., AND TABRIZ, P. Denial of service or denial of security? How attacks on reliability can compromise anonymity. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security* (New York, NY, USA, October 2007), ACM, pp. 92–102.
- [3] CHAUM, D. L. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM* 24, 2 (February 1981), 84–90.
- [4] DAI, W. Two attacks against freedom. <http://www.weidai.com/freedom-attacks.txt>, 2000.
- [5] DANEZIS, G., DINGLEDINE, R., AND MATHEWSON, N. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy* (May 2003), pp. 2–15.

- [6] DANIEL STENBERG, E. A. libcurl, 1998–2009. Open Source C-based multi-platform file transfer library.
- [7] DESMEDT, Y., AND KUROSAWA, K. How to break a practical MIX and design a new one. In *Advances in Cryptology — Eurocrypt 2000, Proceedings* (2000), Springer-Verlag, LNCS 1807, pp. 557–572.
- [8] DIAZ, C., AND SERJANTOV, A. Generalising mixes. In *Proceedings of Privacy Enhancing Technologies workshop (PET 2003)* (March 2003), R. Dingledine, Ed., Springer-Verlag, LNCS 2760, pp. 18–31.
- [9] DINGLEDINE, R. Tor proposal 110: Avoiding infinite length circuits. <https://svn.torproject.org/svn/tor/trunk/doc/spec/proposals/110-avoid-infinite-circuits.txt>, March 2007.
- [10] DINGLEDINE, R. Tor bridges specification. Tech. rep., The Tor Project, <https://svn.torproject.org/svn/tor/trunk/doc/spec/bridges-spec.txt>, 2008.
- [11] DINGLEDINE, R., AND MATHEWSON, N. Design of a blocking-resistant anonymity system. Tech. rep., The Tor Project, <https://svn.torproject.org/svn/tor/trunk/doc/design-paper/blocking.pdf>, 2007.
- [12] DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium* (August 2004).
- [13] FIELDING, R., GETTYS, J., MOGUL, J., FRYSTYK, H., MASINTER, L., LEACH, P., AND BERNERS-LEE, T. *RFC 2616: Hypertext Transfer Protocol — HTTP/1.1*. The Internet Society, June 1999.
- [14] FREEDMAN, M. J., AND MORRIS, R. Tarzan: a peer-to-peer anonymizing network layer. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security* (New York, NY, USA, November 2002), ACM, pp. 193–206.
- [15] FREEDMAN, M. J., SIT, E., CATES, J., AND MORRIS, R. Introducing tarzan, a peer-to-peer anonymizing network layer. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems* (London, UK, 2002), Springer-Verlag, pp. 121–129.
- [16] GOLDSCHLAG, D. M., REED, M. G., AND SYVERSON, P. F. Hiding Routing Information. In *Proceedings of Information Hiding: First International Workshop* (May 1996), R. Anderson, Ed., Springer-Verlag, LNCS 1174, pp. 137–150.
- [17] GÜLCÜ, C., AND TSUDIK, G. Mixing E-mail with Babel. In *Proceedings of the Network and Distributed Security Symposium - NDSS '96* (February 1996), IEEE, pp. 2–16.
- [18] HAN, J., AND LIU, Y. Rumor riding: Anonymizing unstructured peer-to-peer systems. In *ICNP '06: Proceedings of the 2006 IEEE International Conference on Network Protocols* (Washington, DC, USA, Nov 2006), IEEE Computer Society, pp. 22–31.
- [19] HOPPER, N., VASSERMAN, E. Y., AND CHAN-TIN, E. How much anonymity does network latency leak? In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security* (New York, NY, USA, October 2007), ACM, pp. 82–91.
- [20] KEIL, F., SCHMIDT, D., ET AL. Privoxy - a privacy enhancing web proxy. <http://www.privoxy.org/>.
- [21] KESDOGAN, D., EGNER, J., AND BÜSCHKES, R. Stop-and-go MIXes: Providing probabilistic anonymity in an open system. In *Proceedings of the Second International Workshop on Information Hiding* (London, UK, 1998), Springer-Verlag, LNCS 1525, pp. 83–98.
- [22] LANDSIEDEL, O., PIMENIDIS, A., WEHRLE, K., NIEDERMAYER, H., AND CARLE, G. Dynamic multipath onion routing in anonymous peer-to-peer overlay networks. *Global Telecommunications Conference, 2007. GLOBECOM '07. IEEE* (Nov. 2007), 64–69.
- [23] LEVINE, B. N., REITER, M. K., WANG, C., AND WRIGHT, M. K. Timing attacks in low-latency mix-based systems. In *Proceedings of Financial Cryptography (FC '04)* (February 2004), A. Juels, Ed., Springer-Verlag, LNCS 3110, pp. 251–265.
- [24] MCLACHLAN, J., AND HOPPER, N. Don't clog the queue! circuit clogging and mitigation in p2p anonymity schemes. In *Financial Cryptography* (2008), G. Tsudik, Ed., vol. 5143 of *Lecture Notes in Computer Science*, Springer, pp. 31–46.
- [25] MÖLLER, U., COTTRELL, L., PALFRADER, P., AND SASAMAN, L. Mixmaster Protocol — Version 2. IETF Internet Draft, December 2004.
- [26] MURDOCH, S. J. *Covert channel vulnerabilities in anonymity systems*. PhD thesis, University of Cambridge, December 2007.
- [27] MURDOCH, S. J., AND DANEZIS, G. Low-cost traffic analysis of Tor. In *SP '05: Proceedings of the 2005 IEEE Symposium on Security and Privacy* (Washington, DC, USA, May 2005), IEEE Computer Society, pp. 183–195.
- [28] NAMBIAR, A., AND WRIGHT, M. Salsa: a structured approach to large-scale anonymity. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security* (New York, NY, USA, October 2006), ACM, pp. 17–26.
- [29] ØVERLIER, L., AND SYVERSON, P. Locating hidden servers. In *SP '06: Proceedings of the 2006 IEEE Symposium on Security and Privacy* (Washington, DC, USA, May 2006), IEEE Computer Society, pp. 100–114.
- [30] PAPPAS, V., ATHANASOPOULOS, E., IOANNIDIS, S., AND MARKATOS, E. P. Compromising anonymity using packet spinning. In *Proceedings of the 11th Information Security Conference (ISC 2008)* (2008), T.-C. Wu, C.-L. Lei, V. Rijmen, and D.-T. Lee, Eds., vol. 5222 of *Lecture Notes in Computer Science*, Springer, pp. 161–174.
- [31] PERRY, M., AND SQUIRES, S. <https://www.torproject.org/torbutton/>, 2009.
- [32] PFITZMANN, A., PFITZMANN, B., AND WAIDNER, M. ISDN-mixes: Untraceable communication with very small bandwidth overhead. In *Proceedings of the GI/ITG Conference on Communication in Distributed Systems* (February 1991), pp. 451–463.
- [33] RENNHARD, M., AND PLATTNER, B. Introducing MorphMix: Peer-to-Peer based Anonymous Internet Usage with Collusion Detection. In *WPES '02: Proceedings of the 2002 ACM workshop on Privacy in the Electronic Society* (New York, NY, USA, November 2002), ACM, pp. 91–102.
- [34] SERJANTOV, A., DINGLEDINE, R., AND SYVERSON, P. From a trickle to a flood: Active attacks on several mix types. In *IH '02: Revised Papers from the 5th International Workshop on Information Hiding* (London, UK, 2003), F. Petitcolas, Ed., Springer-Verlag, LNCS 2578, pp. 36–52.
- [35] SHMATIKOV, V., AND WANG, M.-H. Timing analysis in low-latency mix networks: Attacks and defenses. In *Proceedings of the 11th European Symposium on Research in Computer Security (ESORICS)* (September 2006), pp. 236–252.
- [36] WIANGSRIPANAWAN, R., SUSILO, W., AND SAFAVI-NAINI, R. Design principles for low latency anonymous network systems secure against timing attacks. In *Proceedings of the fifth Australasian symposium on ACSW frontiers (ACSW '07)* (Darlinghurst, Australia, Australia, 2007), Australian Computer Society, Inc, pp. 183–191.