How Low Can You Go: Balancing Performance with Anonymity in Tor

John Geddes[1], Rob Jansen[2], and Nicholas Hopper[1]

[1] University of Minnesota, Minneapolis, MN
[2] U.S. Naval Research Laboratory, Washington, DC

Abstract. Tor is one of the most popular anonymity systems in use today, in part because of its design goal of providing high performance. This has motivated research into performance enhancing modifications to Tor's circuit scheduling, congestion control, and bandwidth allocation mechanisms. This paper investigates the effects of these proposed modifications on attacks that rely on network measurements as a side channel. We introduce a new class of *induced throttling* attacks in this space that exploit performance enhancing mechanisms to artificially throttle a circuit. We show that these attacks can drastically reduce the set of probable entry guards on a circuit, in many cases uniquely identifying the entry guard. Comparing to existing attacks, we find that although most of the performance enhancing modifications improve the accuracy of network measurements, the effectiveness of the attacks is reduced in some cases by making the Tor network more homogeneous. We conclude with an analysis of the total reduction in anonymity that clients face due to each proposed mechanism.

1 Introduction

The Tor [10] network is a widely-used anonymity and censorship-circumvention tool that provides anonymous Internet access to millions of users every day. This anonymity is provided by routing user traffic through a *circuit* of three relays, using layered encryption to prevent any single relay from seeing more than the next and previous links in the circuit, so that a relay may know the origin or destination of a connection, but not both. The Tor network allows users to participate as clients without contributing as relays, to build a greater variety of plausible uses of the network and provide a larger anonymity set.

One of the key design choices of the Tor system is the goal of building a large anonymity set by providing high performance to as many users as possible, while sacrificing some level of protection from large-scale (global) adversaries. For example, Tor does not attempt to protect against an end-to-end correlation attack that certain mix systems try to prevent [7, 14, 25], as they introduce large costs in increased latency making such systems difficult to use. This performance focus has led researchers to investigate a variety of methods to improve performance, such as using different circuit scheduling algorithms [30], better congestion control [2], and throttling high bandwidth clients [13, 22, 26]. Several of these mechanisms have been or will be incorporated into the Tor software as a result of this research.

One overlooked side effect of these improvements, however, is that in some cases improving the performance of users can also improve the performance of

attacks against the Tor network. For example, several attacks have been proposed [12, 17, 24, 27] that rely on measuring the latency or throughput of a Tor circuit to draw inferences about its source and destination. If an algorithm improves the throughput or responsiveness of Tor circuits this can improve the accuracy of the measurements used by these attacks either directly or by averaging a larger sample. Thus it is important to analyze how these modifications to Tor interact with attacks based on network measurements.

In this paper we investigate this interaction. We start by introducing a new class of attacks based on network measurement, which we call *induced throttling* attacks. In these attacks, an adversarial exit node exploits congestion or traffic admission control algorithms to artificially throttle and unthrottle a chosen circuit without directly sending data through the circuit or relay. This leads to a recognizable pattern in other circuits sharing resources with the target circuit, leaking information about the connection between the client and entry guard. We show that there are highly effective induced throttling attacks against most of the proposed scheduling, flow control, and admission control modifications to Tor, allowing an adversary to uniquely identify entry guards in many cases.

We also examine the effect these algorithms have on previous attacks [17, 24] to see if the improvement in performance, and therefore in network measurements, leads to more successful attacks. Through large-scale simulation, we find that for throughput attacks, the improved network measurements are essentially "cancelled out" by the reduced variance in performance provided by these improvements. We also find that nearly all of the proposed improvements increase the effectiveness of latency-based attacks, in many cases leading to a 10% or higher loss in "degree of anonymity."

Finally, we perform a comprehensive analysis of the combined effects of throughput, induced throttling and latency-measurement attacks. We show that using induced throttling, the combined attacks can in many cases uniquely identify the source of a circuit by a likelihood ratio test. These results indicate that flow and admission control algorithms can have considerable impact on the security as well as performance of the Tor network, and new proposals must be evaluated for resistance to induced throttling.

2 Background and Related Work

This section discusses the proposed performance-enhancing algorithms for and attacks against Tor to facilitate an understanding of our work.

2.1 Tor

As previously mentioned, Tor is the most popular anonymity and censorship-circumvention system, currently consisting of roughly 3000 relays and millions of daily users. During the circuit building process, each client chooses the entry relay into the Tor network from a small set of relays (currently three). Every circuit built by the client will begin with one of these *guard relays* in order to prevent passive logging attacks [28, 32]. It is generally considered feasible for an adversary to eventually de-anonymize a client by combining knowledge of a client's guard nodes with other attacks [4, 17].

2.2 Circuit Scheduling

Tor traditionally used a round robin [15] fair queuing algorithm to determine which among the active circuits on an onion-routing connection to send from next. Although the round robin algorithm is still the software default, the network directory authorities are currently distributing configuration options that enable an EWMA-based algorithm. The EWMA algorithm selects the circuit with the lowest exponentially weighted moving average throughput, and was suggested by Tang and Goldberg [30] in order to reduce latency and prioritize performance for low throughput circuits.

2.3 Congestion Control

The high client-to-relay ratio in Tor causes performance problems that have been the focus of a considerable amount of previous research. The main congestion control mechanism used by Tor is an end-to-end window based system, where the exit relay and client use `SENDME` control cells to infer network level congestion. Tor separates data flowing inbound from data flowing outbound,[3] and congestion control mechanisms operate independently on each flow. Each circuit starts with an initial 1000 cell window which is decremented by the source edge node for every cell sent. When the window reaches 0, the source edge stops sending. Upon receiving 100 cells, the receiver edge node returns a `SENDME` cell to the source edge, allowing the source edge to increment its circuit window by 100 and continue sending more cells.

Tor's *end-to-end* congestion control is slow to react to congestion that occurs in the middle of circuits. Therefore, AlSabah *et al.* introduced N23 [2], a link based algorithm that can instead detect and react to congestion on every *link* in the circuit. Similar to the native congestion control mechanism in Tor, each relay in an N23-controlled circuit initializes its credit balance to $N2 + N3$ and decrements it by one for every cell it forwards. After a node has forwarded $N2$ cells, it returns back a flow control cell containing the number of forwarded cells to the backward relay. Upon receiving a flow control cell from the forward relay, the backward relay updates its credit balance to be $N2+N3$ minus the difference in cells it has forwarded and cells the forward relay has forwarded.

2.4 Traffic Admission Control

Guard nodes in Tor have the ability[4] to throttle clients using a basic rate limiter [9]. The algorithm uses a token bucket whose size and refill rate are configurable to enforce a long-term average throughput while allowing short-term data bursts. The intuition behind the algorithm is that a throttling guard node will limit the client's rate of requests for new data, which will lower the outstanding amount of data that exists inside the network at any given time and generally reduce congestion and improve performance.

[3] Throughout this paper, we use *inbound* to indicate the direction toward the client edge of a circuit, and *outbound* to indicate the direction toward the exit relay edge of a circuit. Relatedly, we use *forward* to indicate the direction of the destination of a data flow, and *backward* to indicate the direction of the source of a data flow.

[4] Tor does not currently enable throttling by default.

There have been many proposed uses of and alterations to the approach outlined above, some of which vary the connections that are throttled [1, 22, 26] and others that vary the throttle rate [13, 22, 26]. Of particular interest are algorithms that dynamically utilize the number of client-to-guard (C-G) connections to adjust throttling rates [22]: the *bitsplit* algorithm divides its configured `BandwidthRate` evenly among C-G connections; the *flag* algorithm uses the number of C-G connections to determine the rate over which a client will get flagged as "high throughput" and throttled; and the *threshold* algorithm throttles the loudest fraction of C-G connections.

2.5 Known Attacks

Murdoch and Danezis previously proposed a Tor circuit clogging attack [27] in which the adversary sends data through a circuit in order to cause congestion and change its latency characteristics. The adversary correlates the latency variations of this circuit with those of circuits through other relays in order to identify the likely relays of a target circuit. The attack requires significant bandwidth in order to produce a signal strong enough for correlation, and it has been shown to be no longer effective [12]. There have been numerous variations on this attack, some of which have simple defenses [12, 18] and others that have low success rates [4]. This work does not consider these "general" congestion attacks where the main focus is keeping bandwidth usage small enough to remain practical. Instead, we focus on the feasibility and anonymity effects of new induced throttling attacks introduced by recent performance enhancing algorithm proposals.

Mittal *et al.* recently proposed "stealthy" throughput attacks [24] where an adversary that controls an exit node of a circuit attempts to find its guard relay by using "probe" clients that measure the attainable throughput through each relay.[5] The adversary may then correlate the circuit throughput measured at the exit node with the throughput of each of its probes to find the guard node with high probability. Some of our attacks also utilize probe clients in order to recognize the signal produced once throttling has been induced on a circuit. Hopper*et al.* [17] propose an attack where an adversary injects malicious javascript into a webpage in order to measure the round trip time of a circuit. The adversary may use these measurements to narrow the possible path the target circuit is taking through the network and approximate the geographical location of the client. As our techniques are similar to both of these attacks, we include them in our evaluation in Section 4 and analysis in Section 7.

3 Methodology

The remainder of this paper will focus on the proposed changes to Tor's internal algorithms that aim to reduce congestion or throttle circuits as discussed in Section 2. In particular, we consider three classes of algorithms that have been recently proposed: EWMA circuit scheduling [30]; N23 congestion control [2]; and bitsplit, flag, and threshold throttling [22]. We will also consider an

[5] The attack is active in that an adversary launches it by sending data to relays, but stealthy in that its data streams are indistinguishable from normal client streams.

ideal throttling algorithm that has perfect knowledge of the traffic type of every stream.[6]

3.1 Metrics

We will explore new algorithm-specific attacks we have developed, as well as previously published generic attacks [17, 24], and quantify the extent to which the attacks affect anonymity. In analyzing the algorithms, we can expect them to have one of two effects: the algorithms may improve the effectiveness of statistical attacks by making side channel *throughput* and *latency* measurements more accurate, improving the adversary's ability to de-anonymize the client; or the algorithms may reduce the noise that an adversary uses to eliminate entry guards and clients from the potential candidate set, frustrating the attacks and improving client anonymity. We will use the following metrics to determine the extent to which our attacks affect anonymity:

Percentile The *percentile* for a candidate target T of an attack is defined as the percent of other candidate targets (i.e., members of the anonymity set) with a lower score than T, based on statistical information the attacker uses to score each candidate as the true target. A higher *percentile* for T means there is a greater likelihood that T is the true target. Percentiles allow an attacker to reduce uncertainty by increasing confidence about the true target, or increasing confidence in rejecting candidates unlikely to be the true target.

Degrees of Anonymity In order to measure the actual level of anonymity lost in the attacks for each algorithm, we first analyze reduction in terms of *entropy* [8, 29] and then compute the *degree of anonymity* [8]. Entropy quantifies uncertainty an adversary has about a target, while the degree of anonymity loss provides us with how much total information a system is leaking. We create a reduced anonymity set based on a threshold value determining what entities to include or discard, then for each possible threshold value we calculate the degree of anonymity *loss*. While this may not show the direct implications of anonymity loss on each client, we can determine the best case scenario for a potential adversary by examining the maximum amount of information leakage possible. Perhaps more importantly, it allows us to do cross experimental comparisons to determine the effect that different algorithms have under different attack scenarios.

Client Probability In order to determine the total reduction in anonymity, we consider the probability distribution that clients have before and after an attack. Given a set of relays R, the *a priori* probability that a relay R_i is the guard node G is $P[G = R_i] = \frac{1}{|R|}$. In addition, for each relay R_i and a set of clients C, the *a priori* probability that a client C_i is the victim V is $P[V = C_i|R_j] = \frac{1}{|C|}$. Therefore, we define the probability that any client C_i is the victim as: $P[V = C_i] = \sum_j P[V = C_i|R_j]P[G = R_j]$. For our purposes, an attack can affect this

[6] The algorithm throttles high throughput nodes at a rate of 50 KiB/s and approximates the difftor approach of AlSabah *et al.* [1].

metric in one of two ways: it either will attempt to identify possible entry guards, thus changing the probability distribution $P[G = R_i]$; or it tries to reduce the set of possible clients given that it knows a potential entry guard, in which case the distribution $P[V = C_i|R_j]$ is updated for each relay R_j.

3.2 Experimental Setup and Model

Our experiments will utilize the Shadow simulator [19, 21], an accurate discrete event simulator that runs the real Tor code over a simulated network. Shadow allows us to configure large scale experiments running on network sizes not feasible using traditional distributed network testbeds [5] while offering precise control over network topology, latency, and bandwidth characteristics. Shadow also allows us to: control Tor's circuit creation process; experiment with our attacks in a safe environment; and run repeatable experiments while only modifying the algorithm or attack scenario of interest, resulting in more controlled and accurate evaluations and comparisons.

We developed a model of the Tor network based on work by Jansen *et al.* [20], and use it as the base of each large scale experiment in the following sections. We will discuss necessary changes to the following base configuration as we explore each specific attack scenario: 160 exit relays, 240 nonexit relays, 2375 web clients, 125 bulk clients, 75 small TorPerf clients, 75 medium TorPerf clients, 75 large TorPerf clients, and 400 HTTP servers. The web client downloads a 320 KiB file from one of the randomly selected servers, after which it sleeps for a time between 1 and 60 seconds drawn uniformly at random before starting the next download. The bulk clients repeatedly download a 5 MiB file with no wait time between downloads. Finally, the TorPerf clients only perform one download every 10 minutes, where the small, medium and large clients download 50 KiB, 1 MiB and 5MiB files respectively. This distribution of clients is used to approximate the findings of McCoy *et al.* [23], Chaabane *et al.* [3] and data from Tor [31].

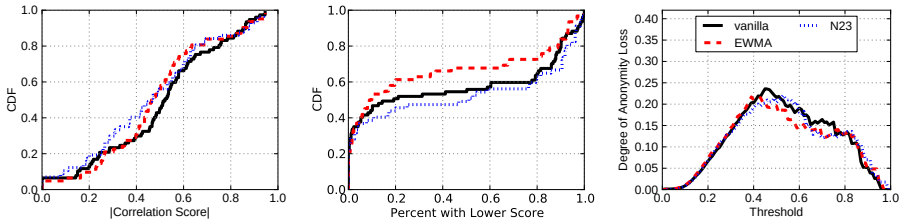
4 Algorithmic Effects on Known Attacks

This section evaluates how recently proposed performance enhancing algorithms affect previously known guard and client identification attacks against Tor.

4.1 Throughput as a Signal

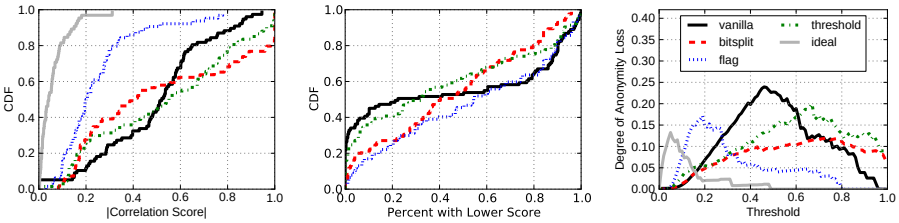
We first explore the scenario of Mittal *et al.* [24], where an attacker is able to identify the guard relay of a circuit with high probability by correlating throughput measured at an adversarial exit node to probe measurements made through a set of entry guards.

We first ran our base experiment from Section 3.2 to discover the circuits that each bulk client created. Then, for every entry G that was not a middle or exit relay for any bulk client, we instantiated a probe client that created a one-hop circuit through G in order to measure its throughput. This was done to minimize the interference of other probes on bulk circuits, where they only potentially affect the circuit they are measuring and no other. We compared vanilla Tor with



(a) Entry Scores (b) Entry Percentile (c) Anonymity Loss

Fig. 1: Results for throughput attack with vanilla Tor compared to EWMA and N23 scheduling and congestion control algorithms.



(a) Entry Scores (b) Entry Percentile (c) Anonymity Loss

Fig. 2: Results for throughput attack with vanilla Tor compared to different throttling algorithms.

6 different algorithms: EWMA circuit scheduling [30], N23 congestion control [2], bitsplit, flag, and threshold throttling [22], and ideal throttling.

The results for the EWMA and N23 algorithms can be seen in Figure 1. Figure 1a shows the correlation of each entry's throughput to that measured by its assigned probe client as a cumulative distribution function (CDF). High correlation scores mean changes in the true entry's throughput strongly corresponds with changes in the probe's throughput. Figure 1b shows, for each candidate entry, the percent of other candidates with a lower score (see Section 3.1). Correlation scores and percentiles can help the attacker reduce uncertainty about the entry node of the target circuit. Figure 1c shows the degree of anonymity loss while varying the threshold value, that is, the minimum correlation score an entry guard must have to be included in set of possible guards. Our results indicate that there are more candidates that match well with the true entry with EWMA (i.e., more uncertainty about the true entry). However, both EWMA and N23 result in insignificant anonymity loss compared to vanilla Tor.

Results for the throttling algorithms are shown in Figure 2. We found incredibly low entry guard correlation scores for the ideal and flag algorithms in Figure 2a, and Figure 2b shows that 48% of entry guards are in the top quintile of the list based on correlation score in vanilla Tor, while only 22-41% of the guards in the throttling algorithm experiments made it in the top quintile. Figure 2c shows a much larger peak in anonymity loss for vanilla Tor compared to all other algorithms, indicating that an adversary would expect more information leakage when choosing the threshold correctly. This implies that the throttling

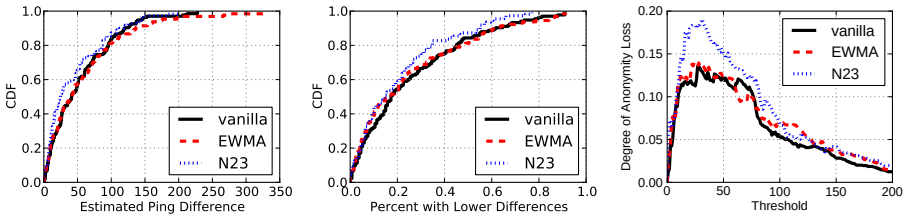
algorithms would actually result in a *larger* anonymity set for the adversary, making the attack less accurate. Intuitively, the throughput of throttled circuits tend to be more similar than the throughput of unthrottled circuits, increasing the uncertainty during the attack. The throttling algorithms effectively smooth out circuit throughput to the configured long-term throttle rate, making it more difficult to distinguish the actual entry guard from the set of potential guards.

4.2 Latency as a Signal

We now explore the latency attack of Hopper *et al.* [17]. They show how an adversarial exit relay, having learned the identity of the entry guard in the circuit, is able to estimate the latency between the client and guard. This is accomplished by creating two ping streams through the circuit, one originating from the client and one from the attacker. The ping stream from the attacker is used to estimate the latency between the entry guard and the exit relay which, when subtracted from the ping times between the client and exit relay produces an estimate of latency between the client and guard. Using network coordinates to compile the set of "actual" latencies between potential clients and guards, the adversary is then able to reduce the anonymity set of clients based on the estimated latency. Since this attack relies on the accuracy of the estimated latency measurements, the majority of the algorithms have the potential to *decrease* the anonymity of the clients by allowing an adversary to discard more potential clients.

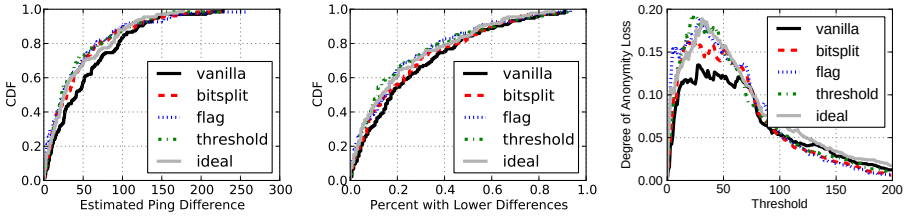
For our experiments, we use the same base configuration with 400 relays and 2500 clients, with an additional 250 victim clients setup to send pings through the Tor circuit every 5 seconds. Then, for each victim client a corresponding attacker client is added, which creates an identical circuit to the one used by the victim, and then sends a ping over this circuit every 5 seconds. These corresponding ping clients are used to calculate the estimated latency between the victim and entry guard as discussed above. In order to determine the actual latencies between the clients and guard node, we utilize the fact that Shadow determines the latency distribution that is sampled from between each node that communicates in the experiment, so we merely assign the median latency of these distributions as the actual latencies between nodes. This would correspond to the analysis done in [17] where it is assumed that these quantities were known *a priori*, so we believe using this "insider information" doesn't contradict any assumptions made in the initial paper outlining the attack. Furthermore, since we're ultimately concerned with how the attacks differ using various algorithms, the analysis should hold.

Similar to the original attack, we take the minimum of all observed ping times seen over both the victim and attacker circuit, denoted T_{VX} and T_{AX} respectively. Then, an estimate of the latency between the victim and entry guard, T_{VE}, is calculated as $\hat{T}_{AE} = T_{VX} - T_{AX} + T_{AE}$, where T_{AE} is the latency between the attacker and entry guard as calculated above. Figures 3a and 4a show the difference in estimated latency computed by an adversary and the actual latency between the client and guard, while Figures 3b and 4b show how these compare with the differences between the estimate and other possible clients.



(a) Ping Differences (b) Percentile (c) Anonymity Loss

Fig. 3: Results of latency attack on EWMA and N23 algorithms.



(a) Ping Differences (b) Percentile (c) Anonymity Loss

Fig. 4: Results of latency attack on the various throttling algorithms.

While these graphs only show a slight improvement for every algorithm except EWMA, the degree of anonymity loss in Figures 3c and 4c shows a noticeable increase in the maximum possible information gain an adversary can achieve. Even though there is only a slight improvement in the accuracy of the latency estimation, this allows an adversary to consider a smaller window around the estimation to filter out potential clients while still retaining similar accuracy rates. This results in a smaller set of potential clients to consider, and thus a higher reduction in anonymity of the victim client.

5 Induced Throttling via Congestion Control

We now look at how an attacker is able to use specific mechanisms in the congestion control algorithms to induce throttling on a target circuit.

5.1 Artificial Congestion

Recall from Section 2.3, the congestion control algorithms send control cells backward to notify edge nodes to send more data. If there is congestion and the nodes go long enough without receiving these cells, they stop sending data until the next control cell is received. Using these mechanisms, an adversarial exit node can implicitly control when a client can and cannot send data forward, thereby inducing artificial congestion.

To demonstrate the effectiveness of such techniques, we introduce a more detailed "torrent" client that models the BitTorrent protocol and mimics a "tit-for-tat" scheme [6]. Instead of downloading a file from a server as is done by the existing web and bulk clients, the torrent client swaps 16 KiB blocks of data with a

peer. This swapping causes large amounts of data to both be uploaded and downloaded and allows us to demonstrate attacks that induce artificial congestion. In our experiment, a bulk and torrent client create connections over the same circuit, where each relay was configured with 128 KiB/s bandwidth. The exit relay would then periodically hold all torrent client control cells in an attempt to throttle the connection. Figure 5 shows the observed throughput of both clients, where the shaded regions indicate periods when the exit relay was holding control cells bound for the torrent client. We can see that approximately 30 seconds into these periods, the torrent client runs out of available cells to send and goes into an idle state, leaving more resources to the bulk client resulting in a rise in the observed throughput.

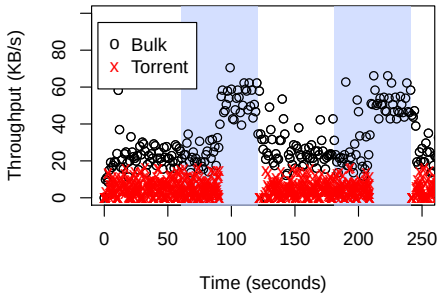
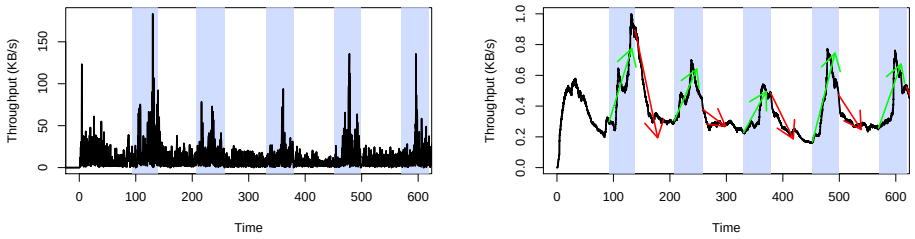


Fig. 5: The effects of using artificial congestion to induce throttling.

Next we want to identify how a potential adversary could utilize this in an attempt to identify entry guards used in a circuit. We can see the intuition behind the attack in Figure 5: when throttling of a circuit is repeatedly toggled, the throughput of all other circuits going through those nodes will increase and then decrease, producing a noticeable pattern which an adversary may be able to detect. We assume a scenario similar to previous attacks [12,24,27], where an adversary controls an exit node and wants to identify the entry guard of a circuit going through them. The adversary creates one-hop probe circuits through possible entry guards by extending circuits through middle relays that the adversary controls, and measures the throughput and congestion on each circuit at a constant interval. The adversary then periodically throttles the circuit by holding control cells and tests for an increase in throughput for the duration of the attack. By repeatedly performing this action an attacker should be able to reduce the possible set of entry guards that the circuit might be using.

5.2 Small Scale Experiment

To test the feasibility of such an attack, we designed a small scale experiment with 20 relays, 190 web clients and 10 bulk clients. One of the exit relays was designated as the adversary and one of the bulk clients was designated as the victim. The victim bulk client was then configured to use the torrent client while creating circuits using the adversary as their exit relay. Then, for each of the 19 remaining non-adversarial relays, a probe client was added and configured to create one-hop circuits through the relay, measuring observed throughput in 100 ms intervals. The adversarial exit relay would then wait until the victim client created the connection, then every 60 seconds would toggle between normal mode in which all control cells are sent as appropriate, and throttle mode where all control cells are held.



(a) Raw Throughput (b) Smoothed Throughput

Fig. 6: Raw and smoothed throughput of probe through guard during attack.

Figure 6a shows the observed throughput at the probe client connected to the entry guard that the victim client was using, where the shaded regions correspond to the periods where the victim client runs out of available cells to send and is throttled. During these periods the probe client sees a large spike in observed throughput as more resources become available at the guard relay. While these changes are visually identifiable, we need a quantitative test that can deal with noise and variability in order to analyze a large number of probe clients and reduce the set of possible guards.

5.3 Smoothing Throughput

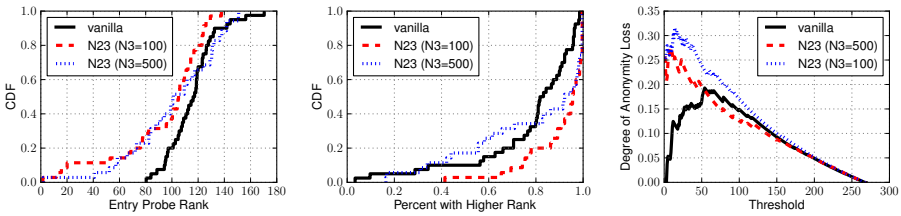
The first step is to smooth out the throughput measurements for each probe client in order to filter out any noise. Given throughput measurements (t_i, b_i), we first compute the exponentially weighted moving average (EWMA), using $\alpha = 0.01$. We then take the output from EWMA and pass it through a cubic spline smoothing algorithm [16] with smoothing parameter $\lambda = 0.5$. The result of this process can be seen in Figure 6b with the normalized smoothed throughput plotted over the shaded attack windows.

5.4 Scoring Algorithm

The intuition behind the scoring algorithm can be seen in Figure 6b. Over each attack window marked by the shaded regions, the guard probe client should see large increases and decreases in throughput at the beginning and end of the window. Here we want the scoring algorithm to place heavy weight on consistent large increases and decreases that align with all the windows, while at the same time minimizing false positives from potentially assigning too much weight to large spikes that randomly happen to align with the attack window.

The first step of the scoring algorithm is to calculate a linear regression on the smoothed throughput over the first δ seconds[7] at the start and end of each attack window and collect the slope values s_i and e_i for the start and end regression. Then for each window i, first sort all probe clients based on their s_i slope value from highest to lowest, and for each probe client record their relative rank r_{s_i}. Repeat this for the slope value e_i, only instead sort the clients from lowest to

[7] Through empirical evaluation we found $\delta = 30$ seconds to be ideal



(a) Entry Average Ranks (b) Percentiles (c) Anonymity Loss

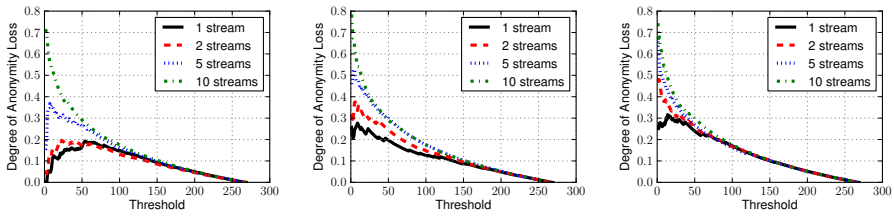
Fig. 7: Large-scale induced throttling via congestion control. The torrent client sends a single stream over the circuit.

highest, again recording their relative rank r_{e_i}. Rankings are used instead of the raw slope value in order to prevent the false positives from large spikes mentioned previously. Now that each probe client has a set of ranks $\{r_{s_1}, r_{e_1}, \ldots, r_{s_n}, r_{e_n}\}$ over n attack windows, the final score assigned to each client is simply the mean of all the ranks, where the lower the score the higher chance the probe client connected through the entry guard.

5.5 Large Scale Experiments

In order to test the accuracy of this attack on a large scale, we used the base experiment setup discussed in Section 3.2, with the addition of one-hop probe clients to connect through each relay. For each run, a random circuit used by a bulk node was chosen to be the attack circuit and the exit node used in the circuit was made an attacker node. The bulk node was then updated to run a torrent client, and the probe clients that were initially set up to probe the middle and exit relays were removed. For each algorithm, we performed 40 runs with the different attack circuits chosen for each run. We configure the experiments with the vanilla Tor algorithm which uses `SENDME` cells for congestion control, and the N23 congestion control algorithm with $N3 = 500$ and $N3 = 100$. We experimented with having the torrent client send 1, 2, 5, and 10 streams over the circuit. The single stream results are shown in Figure 7 and the multiple stream results in Figure 8.

Figure 7a shows the CDF of the average score computed by the ranking algorithm for the entry guards' probe client, while Figure 7b shows the CDF of the percent of probe clients with a higher rank (i.e. worse score) than the true guard's probe client. Interestingly, even though in vanilla Tor not a single entry guard probe had a score better than 150 out of 400, we still see about 80% of the entry guard probe clients were in the 25^{th} percentile amongst all probe clients. Furthermore, we see that the attack is much more successful with the N23 algorithm, especially with $N3 = 100$, with peaks in degree of anonymity loss at 31%, compared to 27% with $N3 = 500$ and 19% in vanilla Tor. This is due to the fact that it is easier to induce throttling with N23, especially when the $N3$ value is low. In vanilla Tor, the initial window is set at 1000 cells, while for the N23 algorithm this would be $N2 + N3$, which works out to 510 and 110 cells for $N3 = 500$ and $N2 = 100$ respectively (default value for $N2$ is 10). The outcome



(a) Vanilla (b) N23 (N3=500) (c) N23 (N3=100)

Fig. 8: Large-scale induced throttling via congestion control. The torrent client sends multiple streams over the circuit. The anonymity loss is higher when the torrent client utilizes more streams.

is that for N23 with $N3 = 100$, we were able to throttle the attack circuit around 12 times, resulting in 24 comparison points over all attack windows, while both with $N3 = 500$ and vanilla Tor we see around 7 attack windows, resulting in 14 comparison points. The reason that we see slightly better entry probe rankings with $N3 = 500$ than vanilla Tor is because with N23, each node buffers cells when it runs out of credit, while vanilla Tor buffers cells at the client. This means when the congestion control cell is finally sent by the attacker, it would cause the entry guard to flush the circuits buffer and cause an immediately noticeable change in throughput and thus a higher rank score for the entry guard.

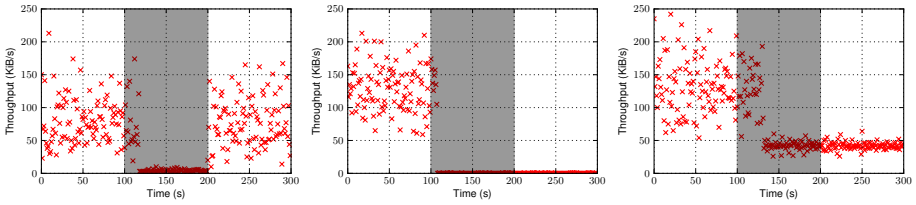
While using one circuit for one stream is the ideal greedy strategy for a BitTorrent client using Tor, it may not always be feasible to accomplish this. To explore what effects sending multiple streams over a circuit has on the attacks, for each algorithm we experimented having our torrent client send 2, 5, and 10 streams over the circuit that the attacker throttles. The results are shown in Figure 8. For each algorithm, we can see how the degree of anonymity loss changes as more streams are multiplexed over a single Tor circuit. Not surprisingly, all three algorithms have a high degree of anonymity loss when sending 10 streams over the circuit, as this dramatically increases the amount of data being sent over the circuit. Thus, when the artificial throttling is induced, there will be larger variations and changes in observed throughput at the entry guard's probe client. Even adding a single extra stream to the circuit can in some cases cause a noticeable reduction in the degree of anonymity, as particularly exemplified by the N23 algorithm with $N3 = 100$ (Figure 8c).

6 Induced Throttling via Traffic Admission Control

We now explore how an attacker is able to use specific mechanisms in proposed traffic admission control algorithms to induce throttling on a target circuit, creating a throughput signal at much lower cost than the techniques required in [24].

6.1 Connection Sybils

Recall that each of the algorithms proposed in [22] relies on the number of client-to-guard connections to adaptively adjust the throttle rate (see Section 2). Unfortunately, this feature may be controlled by the adversary during an active



(a) Bitsplit (b) Flag (c) Threshold

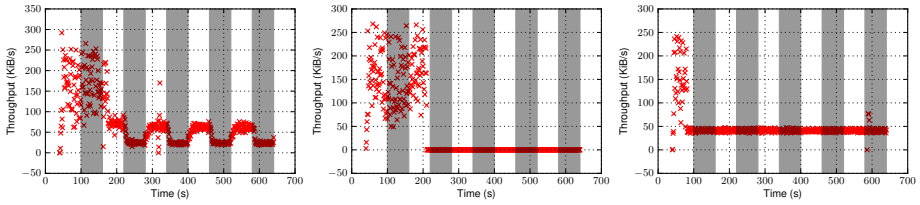
Fig. 9: Small scale sybil attack on the bandwidth throttling algorithms from [22]. The shaded area represents the period during which the attack is active. The sybils cause an easily recognizable drop in throughput on the target circuit.

sybil attack [11]: an adversary may either modify a Tor client to create multiple connections to a target guard relay instead of just one, or boot multiple Tor clients that are each instructed to connect to the same target.[8] As a result, the number of connections C at the target will increase to $C = C_n + C_s$, where C_n is the number of normal connections and C_s is the number of sybil connections. The throttling algorithms will be affected as follows: the *bitsplit* algorithm will lower the throttle rate to $\frac{\texttt{BandwidthRate}}{C_n+C_s}$; the *flag* algorithm will throttle any connection whose average throughput has ever exceeded $\frac{\texttt{BandwidthRate}}{C_n+C_s}$ to the configured flag rate; and the *threshold* algorithm will throttle the loudest fraction f of connections to the throughput of the quietest of that throttled set (but no less than a floor of 50 KiB/s).[9] Therefore, the attacker may cause the throttle rate at any guard relay to reduce dramatically in all of the algorithms by using enough sybils. In this way, an adversary controlling an exit node can determine the guard node belonging to a target circuit with high probability. Note that the attacker need not use any bandwidth or computational resources beyond that which is required to establish the connections from its client(s) to the target guard relay.

We test the feasibility of this attack in Shadow. We configure a network with 5 relays, a file server, and a single victim client that downloads a large file from the server through Tor for 300 seconds. The adversary controls the exit relay on the client's circuit and therefore is able to compute the client's throughput. The attacker starts multiple sybil nodes at time $t = 100$ seconds that each connect to the same entry relay used by the victim. The sybil nodes are shut down at $t = 200$, after being active for 100 seconds.

The results of this attack on each algorithm are shown in Figure 9, where the shaded area represents the time during which the attack was active. Figure 9a shows that the *bitsplit* algorithm is only affected while the attack is active, after which the client throughput returns to normal. However, the client throughput remains degraded in both Figures 9b and 9c—the *flag* algorithm flagged the client as a high throughput node and did not unflag it while the *threshold* al-

[8] A similar attack was previously described in [22], Section 5.2, Attack 4.
[9] We use a flag rate of 5 KiB/s and a threshold of $f = 0.10$ as advised in [22]



(a) Bitsplit (b) Flag (c) Threshold

Fig. 10: Large scale sybil attack on the bandwidth throttling algorithms from [22]. The shaded area represents the period during which the attack is active. Due to token bucket sizes, the throughput signal may be missed if the attack phases are too short.

gorithm continued to throttle at the 50 KiB/s floor. Further, notice that the throttling does not occur until roughly 10 to 20 seconds after the attack has begun. This is due to the size of the token bucket, i.e., the `BandwidthBurst` configuration: the attack causes the refill rate to drop dramatically, but it takes time for the client to use up the existing tokens in the bucket. In addition to the delay associated with the token bucket size, Figure 9c shows added delay in the *threshold* algorithm because it only updates throttle rates once per minute.

6.2 Large Scale Experiments

We further explore the sybil attack on a large scale in Shadow. In addition to the base experiment setup discussed in Section 3.2, we add a victim client who downloads a large file through a circuit with an adversarial exit and a known target guard. The adversary starts sybil nodes and instructs them to connect to the known target guard at time $t = 100$. The sybil nodes cycle through 60 second active and inactive phases, and the adversary measures the throughput of the circuit.

The results of this attack on each algorithm are shown in Figure 10, where the shaded area again represents the time during which the attack was active. In the large scale experiment, only the *bitsplit* algorithm (Figure 10a) produced a repeatable signal while the throttle rate remained constant after the first phase for both *flag* (Figure 10b) and *threshold* (Figure 10c). Also, these results show the importance of correctly computing the attack duration: the signal was missed during the first phase for both *bitsplit* and *flag* because the token bucket had not yet fully drained.

6.3 Search Extensions

Because the drop in throughput during the attack is easily recognizable, it is much easier to carry out than those discussed in Section 5. Therefore, in addition to statistical correlations that eliminate potential guards from a candidate set for a given circuit, a *search strategy* over the potential guard set will also be useful. In a linear search, the adversary would attack each candidate guard one by one until the throughput signal on the target circuit is recognized. This strategy is simple, but it may take a long time to test every candidate. A binary search, where the attacker tests half of the candidates in each phase of the search, would

significantly reduce the search time. Note that a binary search may be ineffective on certain configurations of the *flag* and *threshold* algorithms because of the lack of a repeatable signal (see Figures 9 and 10).

Regardless of the search strategy, a successful attack will contain enough sybils to allow the adversary to recognize the throughput signal, but otherwise be as fast as possible. Given our results above, the attack should consider the token bucket size and refill rate of each candidate guard to aid in determining the number of sybils to launch and the length of time each sybil should remain active. An adversary who controls the circuit exit may compute the average circuit throughput and estimate the amount of time it would take for the circuit to deplete the remaining tokens in a target guard's bucket during an attack. Each sybil should then remain active for at least that amount of time. Figure 10 shows that the throughput signal may be missed without these considerations.

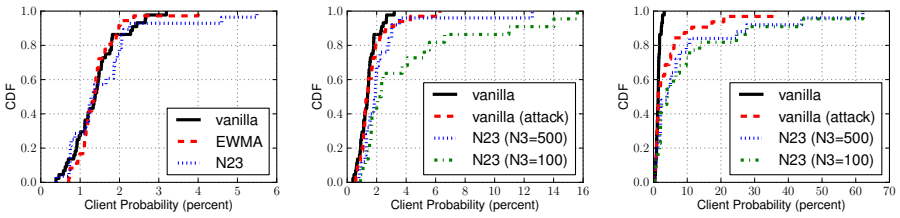
7 Analysis

Having seen how the algorithms perform independently in different attack scenarios, we now want to examine the overall effects on anonymity that each algorithm has. For our analysis, we use client probability distributions as discussed in Section 3.1 to measure how much information is gained by an adversary. Recall that we have the probability that a client is the victim being $P[V = C_i] = \sum_j P[V = C_i|R_j]P[G = R_j]$ for a set of relays R and clients C. Now we need to determine how to update the probability distributions for $P[G = R_j]$ and $P[V = C_i|R_j]$ based on the attacks we've covered.

There are three attacks discussed that are used to learn information about the guard node used in a circuit, determining $P[G = R_j]$. The throughput attack and artificial throttling attack both attempt to reduce the set of possible entry guards by assigning a score to each guard and using a threshold value to determine the reduced set. For each attack, we compile the set of scores assigned to the *actual* guard nodes, and use this set as input to a kernel density estimator in order to generate an estimate of the probability density function, $\hat{P}$. Then, given a relay R_j with a score $score(R_j)$ we can compute $P[G = R_j] = \hat{P}[X = score(R_j)]$. For the sybil attacks on the throttling algorithms we were able to uniquely identify the entry guard, so we have $P[G = R_j] = 1$ if R_j is the guard, otherwise it's 0. Therefore, denoting R_G as the guard relay, we have $P[V = C_i] = P[V = C_i|R_G]$.

For determining the probability distribution for $P[V = C_i|R_j]$, recall that the latency attack computes the difference between the estimated latency $\hat{T}_{VE}$ and the actual latency T_{VE} as the score, and ranks potential clients that way. Using the absolute value of the difference as the score, we compute the probability density function $\hat{P}$ in the same way as we did for $P[G = R_j]$. Therefore, to compute $P[V = C_i|R_j]$, we let $lat_{C_iR_j}$ be the actual latency between client C_i and relay R_j, and $\hat{T}_{VE}$ be the estimate latency between the client and entry guard. Then, with $diff = |lat_{C_iR_j} - \hat{T}_{VE}|$ we have $P[V = C_i|R_j] = \hat{P}[X = diff]$ from the computed probability density function.

Our analysis first concentrates on how the algorithms perform with the throughput and latency attack compared to vanilla Tor. We then focus on the



(a) Throughput (b) Induced Throttling (c) Perfect Knowledge

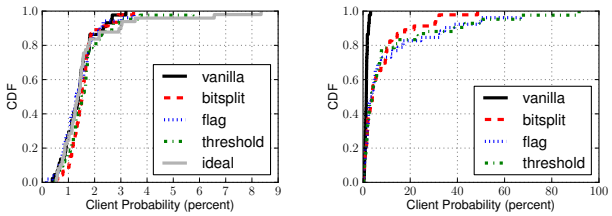
Fig. 11: Victim client probabilities, various congestion control attack scenarios.

new induced throttling attacks shown in Section 5 and 6 to see if there are improvements over either vanilla Tor or the throughput attack with the algorithms. For each attack we use a set of potential victims $\{V_i\}$ and their entry guards G_i and compute the probabilities $P[V = V_i]$ as shown above.

The results for the algorithms EWMA, N23, and the induced throttling attacks described in Section 5 are shown in Figure 11. We can see in Figure 11a that with just the throughput and latency attack, vanilla Tor leaks about the same amount of anonymity of a client as EWMA and N23. The exception to this is that a tiny proportion of clients have probabilities ranging from 4-6% which is outside the range in vanilla Tor. Referring back to Figure 3 we see that these algorithms, N23 in particular, leak slightly more information than vanilla Tor based on latency estimation. While sometimes this information gain might be counteracted by the amount of possible entry guards that need to be considered, there are a small amount of cases where the guard set is reduced enough that the extra information from the latency attack translates into a higher probability for the client.

When replacing the throughput attack with the induced throttling attack we start to see a larger divergence in client probabilities, as shown in Figure 11b. While the induced throttling attack with vanilla Tor leaks slightly more information than vanilla Tor with the throughput attack, N23 with $N3 = 500$ has higher client probabilities than both attacks on vanilla Tor and higher than N23 with just the throughput attack. Furthermore, N23 with $N3 = 100$ does significantly better than all previous algorithms, leaking more information than vanilla Tor for almost half the clients, reaching probabilities as high as 15%.

The results in Figure 11b assume that the client only sends one stream over the circuit, the worst case scenario for an adversary. As shown in Figure 8, as the number streams multiplexed over the circuit increases, the degree of anonymity loss sharply approaches 100% implying that an adversary would be able to uniquely identify the entry guard. An analysis with this assumption of "perfect knowledge" can be seen in Figure 11c, where $P[G = R_j] = 1$ when R_j is the entry guard. Here we see a dramatic improvement from when a client only sends a single stream over the circuit, with some clients having probabilities as high as 60%, compared to a peak of 15% with a single stream.



(a) Throughput (b) Induced Throttling

Fig. 12: Victim client probabilities, various traffic admission control attack scenarios.

Using the throughput attack with the traffic admission control algorithms produces similar results as N23 and EWMA, as shown in Figure 12a. There is a slightly higher upper bound in the client probability caused by the threshold and ideal throttling algorithms, but for the most part these results line up fairly closely to what was previously seen. Given that the throttling algorithms all had similar peaks to N23 with respect to the loss of anonymity in the latency attacks, these results aren't too surprising. Even with the improved performance of the latency attack, these gains are wiped out by the fact that the throughput attack results in too many guards that need to be considered in relation to the clients. However, when using the sybil attack to induce throttling under the assumption that the adversary is able to uniquely identify the entry guard in use, we see dramatically higher client probabilities. Figure 12b shows the result of this analysis, where at the extreme end we see clients with probabilities as high as 90%. This is due to the fact that with the sybil attack we are able to identify the exact entry guard used by each victim, thus reducing the noise from having to consider the latency of clients based on other possible relays. This very effectively demonstrates the level of anonymity lost when an adversary is able to significantly reduce the set of candidate entry guards.

8 Conclusion

While high performance is vital to the Tor system, algorithms which seek to improve allocation of network resources via more advanced congestion control or traffic admission algorithms need to take into account the implications on anonymity, both with respect to existing attacks and the potential for new ones. To this effect, we introduce a new class of *induced throttling* attacks and demonstrate the effectiveness across a wide variety of performance enhancing algorithms, resulting in dramatic information leakage on victim clients. Using the new class of attacks, we perform a comprehensive analysis on the implications on anonymity, showing both the effects the algorithms have on existing attacks, as well as showing the increase in information gain from the new attacks.

Preventing these new attacks isn't straightforward, as in many cases the adversary is merely exploiting the underlying mechanisms in the algorithms. With the *induced throttling* attacks on vanilla and N23 congestion control, an adversary acts exactly as they should under heavy congestion, so prevention or detection becomes difficult without completely changing the algorithm. In these

cases it comes down to the performance/anonymity trade-off. However, in the throttling algorithms the adversary is taking advantage of the fact that only the raw number of open connections are considered when calculating the throttling rate, allowing Sybil connections to be created using negligible resources. A throttling algorithm might prevent this by considering only *active* connections which have seen a minimum amount of bandwidth over a certain time period, forcing the attacker to spend a non-trivial amount of resources to significantly affect the throttle rate. The throttling rate could also be weighted by each connection's average bandwidth, creating a direct correlation between the bandwidth an adversary must provide and its influence on the throttling rate. Alternatively, throttling algorithms that do not directly consider the number of connections would not be vulnerable to the attacks in this paper.

Acknowledgments We would like to thank our shepherds Roger Dingledine and Damon McCoy and the anonymous reviewers for their comments that helped improve this paper. This work was supported by NSF grant 0915145.

References

1. AlSabah, M., Bauer, K., Goldberg, I.: Enhancing Tor's performance using real-time traffic classification. In: Proceedings of the 2012 ACM conference on Computer and communications security, ACM (2012)
2. AlSabah, M., Bauer, K., Goldberg, I., Grunwald, D., McCoy, D., Savage, S., Voelker, G.M.: DefenestraTor: Throwing out windows in Tor. In: Privacy Enhancing Technologies, Springer (2011)
3. Chaabane, A., Manils, P., Kaafar, M.A.: Digging into anonymous traffic: A deep analysis of the tor anonymizing network. In: Network and System Security (NSS), 2010 4th International Conference on. (2010)
4. Chakravarty, S., Stavrou, A., Keromytis, A.D.: Traffic Analysis Against Low-Latency Anonymity Networks Using Available Bandwidth Estimation. In: Proceedings of the European Symposium Research Computer Security - ESORICS'10. (2010)
5. Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M., Bowman, M.: PlanetLab: an overlay testbed for broad-coverage services. SIGCOMM Computer Communication Review **33** (2003)
6. Cohen, B.: Incentives build robustness in BitTorrent. In: Workshop on Economics of Peer-to-Peer systems. Volume 6. (2003)
7. Danezis, G., Dingledine, R., Mathewson, N.: Mixminion: Design of a type III anonymous remailer protocol. In: Proc. of IEEE Security and Privacy. (2003)
8. Diaz, C., Seys, S., Claessens, J., Preneel, B.: Towards measuring anonymity. In: Privacy Enhancing Technologies, Springer (2003)
9. Dingledine, R.: Adaptive throttling of Tor clients by entry guards. Technical Report 2010-09-001, The Tor Project. (September 2010)
10. Dingledine, R., Mathewson, N., Syverson, P.: Tor: The Second-Generation Onion Router. In: In Proceedings of the 13th Usenix Security Symposium. (2004)
11. Douceur, J.R.: The Sybil Attack. In: IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems. (2002)
12. Evans, N.S., Dingledine, R., Grothoff, C.: A practical congestion attack on Tor using long paths. In: Proceedings of the 18th USENIX Security Symposium. (2009)

13. Gopal, D., Heninger, N.: Torchestra: Reducing interactive traffic delays over Tor. In: Proc. of the Workshop on Privacy in the Electronic Society. (2012)
14. Gulcu, C., Tsudik, G.: Mixing E-mail with Babel. In: Proceedings of the Symposium on Network and Distributed System Security. (1996)
15. Hahne, E.: Round-robin scheduling for max-min fairness in data networks. IEEE Journal on Selected Areas in Communications **9**(7) (1991)
16. Hastie, T.J., Tibshirani, R.J.: Generalized additive models. Volume 43. (1990)
17. Hopper, N., Vasserman, E.Y., Chan-Tin, E.: How much anonymity does network latency leak? In: Proceedings of the 14th ACM conference on Computer and communications security, ACM (2007)
18. Houmansadr, A., Borisov, N.: SWIRL: A Scalable Watermark to Detect Correlated Network Flows. In: Proc. of the Network and Distributed Security Symp. (2011)
19. Jansen, R.: The Shadow Simulator. http://shadow.cs.umn.edu/
20. Jansen, R., Bauer, K., Hopper, N., Dingledine, R.: Methodically Modeling the Tor Network. In: Proceedings of the 5th Workshop on Cyber Security Experimentation and Test. (August 2012)
21. Jansen, R., Hopper, N.: Shadow: Running Tor in a Box for Accurate and Efficient Experimentation. In: Proceedings of the 19th Network and Distributed System Security Symposium. (2012)
22. Jansen, R., Syverson, P., Hopper, N.: Throttling Tor Bandwidth Parasites. In: Proceedings of the 21st USENIX Security Symposium. (2012)
23. McCoy, D., Bauer, K., Grunwald, D., Kohno, T., Sicker, D.: Shining light in dark places: Understanding the Tor network. In: Privacy Enhancing Technologies, Springer (2008)
24. Mittal, P., Khurshid, A., Juen, J., Caesar, M., Borisov, N.: Stealthy traffic analysis of low-latency anonymous communication using throughput fingerprinting. In: Proceedings of the 18th ACM conference on Computer and communications security, ACM (2011)
25. Möller, U., Cottrell, L., Palfrader, P., Sassaman, L.: Mixmaster protocol version 2. Draft, July (2003)
26. Moore, W.B., Wacek, C., Sherr, M.: Exploring the Potential Benefits of Expanded Rate Limiting in Tor: Slow and Steady Wins the Race With Tortoise. In: Proceedings of 2011 Annual Computer Security Applications Conference. (2011)
27. Murdoch, S.J., Danezis, G.: Low-cost traffic analysis of Tor. In: Security and Privacy, 2005 IEEE Symposium on, IEEE (2005)
28. Øverlier, L., Syverson, P.: Locating Hidden Servers. In: Proceedings of the 2006 IEEE Symposium on Security and Privacy. (2006)
29. Serjantov, A., Danezis, G.: Towards an information theoretic metric for anonymity. In: Privacy Enhancing Technologies, Springer (2003)
30. Tang, C., Goldberg, I.: An improved algorithm for Tor circuit scheduling. In: Proceedings of the 17th ACM conference on Computer and communications security, ACM (2010)
31. The Tor Project: The Tor Metrics Portal. https://metrics.torproject.org/
32. Wright, M., Adler, M., Levine, B.N., Shields, C.: Defending Anonymous Communication Against Passive Logging Attacks. In: Proceedings of the 2003 IEEE Symposium on Security and Privacy. (May 2003)