mTor: A Multipath Tor Routing Beyond Bandwidth Throttling

Lei Yang and Fengjun Li
The University of Kansas, Lawrence, Kansas, US, 66045
Email: {lei.yang, fli}@ku.edu

***Abstract*—One of the main obstacles that impede further expansion of Tor, the most popular anonymous communication system, is its large performance variance. The problem becomes worse when bandwidth-intensive applications, such as video streaming, contend with latency-sensitive applications, such as web browsing, for the scarce resources. Most of the existing solutions involve circuit-scheduling techniques to prioritize interactive traffic over bulk traffic or to completely throttle traffic of bandwidth-intensive applications. However, these approaches not only rely on accurate detection of traffic types but also adopt detection strategies that are easy to game. In this paper, we propose a different approach by exploring new capabilities of Tor to support bulk data transfers without degrading the performance of interactive traffic. Based on our observations that a large portion of low-bandwidth relays are under-utilized, we develop a multi-path Tor (*m*Tor) routing algorithm to cater to bandwidth-intensive applications by constructing multiple circuits across low-bandwidth Tor relays. We present a self-adaptive "pulling" scheduling technique to dynamically allocate cells across multiple circuits, and an active congestion detection scheme to prevent slow circuits from becoming a bottleneck of the entire tunnel. Based on the results from experiments on the live Tor network and simulations over the Shadow simulator [1], we conclude that *m*Tor not only achieves a desirable performance for bandwidth-intensive applications by utilizing multiple low-bandwidth relays, but also benefits latency-sensitive applications by reducing the load on high-bandwidth relays.**

I. Introduction

Tor [2], also known as The Second-Generation Onion Router, is the most popular and widely deployed tool for anonymous communications over the Internet today. It provides services to millions of users on a daily basis [3]. Tor was initially designed to provide anonymity service with low latency and reasonable throughput for interactive applications such as web browsing, instant message and SSH. However, Tor users often experience performance with large variance, which becomes a big obstacle to impede Tor's further expansion.

Many causes lead to Tor's performance variance. Two main reasons that slow down Tor are: (1) the path selection schemes designed for balancing traffic over the entire Tor network [4], [5], and (2) bulk traffic contending for bandwidth with interactive users [6]–[8]. The current path selection scheme excludes slowest relays while selecting from the remaining relays in proportion to their weighted bandwidth (i.e., consensus weight). In other words, a router with higher weighted bandwidth is more likely to be selected. The preference on high-bandwidth relays in the current path selection scheme makes some routers persist in being under-utilized or congested [5]. On the other hand, bulk traffic also significantly affects the performance under the current Tor architecture. McCoy et al. found that 40% of network capacity is consumed by only a small fraction of BitTorrent users (3.5% of all connections) [6]. To achieve a good performance, most of BitTorrent traffic go through the scarce high-bandwidth relays, which increases the congestion and degrades the performance of interactive users.

To solve Tor's performance problems for interactive traffic, the existing work follows three thrusts. The first attempt is to increase the overall network capacity. [9]–[12] proposed various incentive schemes to attract more people in contributing more Tor relays. The increased overall bandwidth benefits both interactive and bulk data clients. However, we argue that without an effective allocation mechanism, the newly introduced resources may attract more bulk traffic requests that consume bandwidth more aggressively than interactive traffic and thus make the situation worse. Along the second direction are the schemes aiming at optimizing path selection based on relays' geographic locations [13] or performance history [4], [5]. A better performance is achieved for interactive users by avoiding instinctively slow relay and/or transiently congested relays being selected in the anonymity route. However, while optimizing the performance of individual circuits, these approaches reduce the global utility of the anonymity network by imposing too many restrains that excludes the capacity of low-bandwidth relays despite very limited. A different attempt in improving the performance of interactive traffic is to prioritize it over bulk traffic [7] or completely throttle bulk traffic [8]. However, one problem with these schemes is that the metrics used to detect bulk traffic are simple to game [14]. Besides, the throttling techniques enforce strict restrictions to block bulk traffic resulting in reduced overall network utilization.

Moreover, we argue that simply blocking bandwidth-intensive applications (e.g., video chat and video streaming) over Tor is not a good solution due to their increasing popularity. Cisco forecasts that the portion of video traffic will increase from 48% of total traffic in 2012 to 67% in 2017 [15]. In the mean time, the traffic from web, email, and data applications is expected to decrease from 23% to 18% [16]. On the other hand, the demand for anonymity services is continuously increasing in response to the increased network monitoring and censorship. For instance, several video chat applications (e.g., *Gajim* and *Riseup Chat*) recommend users with strong anonymity needs to use their service over Tor. It is reasonable to conjecture that the increasing demand for privacy protection among Internet users will generate a large number of bulk data

978-1-4673-7876-5/15/$31.00 ©2015 IEEE

requests to the Tor network. Therefore, supporting bandwidth-intensive applications on existing anonymous communication platforms is of great necessity.

In this work, we propose a multipath Tor (*m*Tor) routing scheme for bandwidth-intensive applications by utilizing low-bandwidth relays, without degrading the performance of latency-sensitive applications. In particular, *m*Tor selects relays from a set of low-bandwidth routers, which are under-utilized or completely excluded by the current path selection algorithms, to construct multiple circuits and form an anonymous *tunnel* for bulk data transfer. *m*Tor adopts a self-adaptive "pulling" scheduling technique to dynamically allocate cells across multiple circuits, and an active congestion detection scheme to prevent slow circuits from becoming a bottleneck of the entire tunnel. We conduct experiments on live Tor and find that by increasing the number of circuits in a tunnel, *m*Tor can always achieve a desirable performance for bulk data transfer. Our simulation results on the Shadow simulator show that *m*Tor significantly reduces the load on high-bandwidth relays and potentially benefits interactive streams.

To this end, this paper makes the following contributions: (1) We investigate the significant performance variance of different types of relays and the low utilization of low-bandwidth relays caused by the current path selection algorithm and explore the reasons that motivate us to design a multipath Tor. (2) We elaborate the detailed designs of *m*Tor that utilizes low-bandwidth relays and dynamically distributes traffic across multiple circuits. (3) We evaluate the performance of *m*Tor on both live Tor platform and the Shadow simulator. And (4) We analyze the security of *m*Tor against traffic correlation attacks and theoretically measure the anonymity it provides.

II. Background

Tor and Tor path selection. The Tor network is an overlay network contributed by volunteers running Onion Routers (ORs). An OR's parameters, such as bandwidth and exit policy, are configured by its operator, and sent to a set of centralized servers known as directory authorities. These authorities also actively measure the real capacity of ORs, negotiate the network status and publish a consensus about all ORs. Onion proxy (OP) running at the clients will download the consensus file, and select ORs in proportion to their weighted bandwidth to establish a circuit through them. OP encrypts the application data in layers, packs them to 512-byte cells and sends data cells through the circuit. Each relay along the circuit decrypts its layer and forwards the cell to the next relay until it reaches the last relay (known as "exit"), which further forwards the data to the original destination. Each hop only knows who has sent the data (predecessor) and to whom it is relaying (successor) due to layered encryption. OP typically selects the first-hop router from a set of "entry guards" [17] to defend against the predecessor attack [18].

Flow control. Tor uses a two-layer window-based end-to-end flow control scheme to guarantee a steady flow between the client and the exit. Since multiple streams multiplex a circuit, the outer layer is circuit-level control which restricts the number of cells that can be transmitted on a circuit for all streams. The inner layer is a stream-level control applied to individual streams. At two edges of a circuit, OP and exit control the speed of data cells entering and leaving the circuit by keeping track of circuit and stream windows. By default, a circuit window starts with 1000 cells and a stream window is initialized to 500 cells. When a data cell is sent, both windows will be decreased by one. When a stream window is decreased to zero, the sender stops sending from this stream. When a circuit window reaches zero, the sender stops sending from all streams on this circuit. Windows are increased when the corresponding acknowledgment cell known as *SENDME* is received. For every 100 cells received on a circuit, the receiver sends a circuit *SENDME* to inform the sender to forward another 100 cells from this circuit. For every 50 cells received from a stream in this circuit, the receiver sends a stream *SENDME* to request another 50 cells from this stream.

III. Why Tor is Possible for Bulk Applications

As an overlay network, bandwidth is a scarce resource for Tor users. If all bandwidth had been fully utilized, there is no doubt that support to bulk data applications will hurt all current users. So, in this section the first question that we will explore is the current utilization of Tor relays. Then, if some routers are under-utilized, we will examine what quality of services they can provide. Finally, we will discuss the observations that motivate us to design mTor.

A. Low Utilization of Low-BW Relays

Since there is no publicly available data about relay utilization, to answer the first question, we implemented a path selection simulator which constructs a huge volume of circuits using the default Tor path selection scheme according to a given network consensus. A relay is considered as an idle relay if it is not used by any circuit. These idle relays can be potentially utilized to serve bulk traffic. To avoid introducing extra burden to the Tor network, each circuit construction terminates once the three hops are selected without further key exchange and TLS connection establishment.

The goal is to explore how the number of concurrently constructed circuits affects the number of idle relays. Since the statistics for the number of users per hour on the live Tor are considered too detailed and might put users at risk, Tor Project publishes only aggregated statistics for a 24-hour period. We get the daily statistics between 2011 and 2015 from [3], the number of daily visits changes from 0.49M to 5.92M with a mean of 1.8M and a median of 0.99M. The recent daily statistics in 2015 are around 2.2M, which leads to an estimated average client visit of 92K per hour, so we compare the results of constructing 50K, 100K, 150K and 200K concurrent circuits.

Results. Figure 1 shows the relation between the number of established circuits and the number of idle relays. It is obvious that the more clients using Tor at the same time, the fewer relays remaining idle. But we can also see that even if 200K clients are simultaneously building circuits, 25.6% of relays are still unused, indicating that many relays with available bandwidth are less utilized by the current path selection scheme.

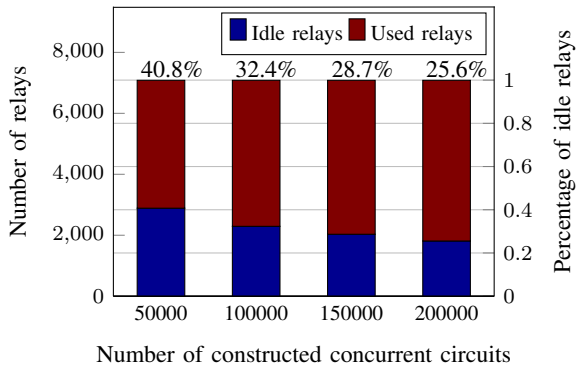


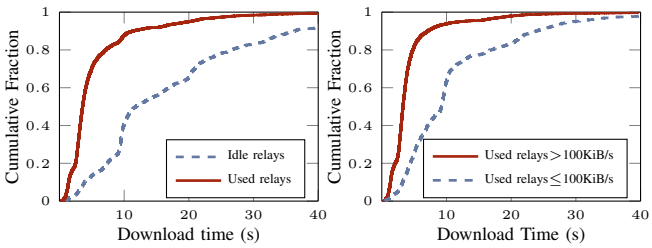
Fig. 1: The number of idle relays changes with the number of constructed circuits.

We further explore the characteristic of relays that constitute the set of idle relays. When 200K circuits are constructed, 1,811 relays with bandwidth varying from 1KiB/s to 7,400KiB/s are idle. Among them, the bandwidth of 1,769 relays (97.7%) is less than 100 KiB/s. Tor assigns "Fast" flag to relays with bandwidth either in the top $7/8^{th}$ active routers or at least 100KiB/s. By default, OP only selects fast relays to construct circuits. In this work, we define low-bandwidth relays as the relays with less than 100KiB/s. There are totally 2,740 relays with bandwidth of less than 100 KiB/s in our given consensus, while almost 64.6% of them are idle. From this experiment, we answer the first question: there are a large number of under-utilized relays in Tor and the bandwidth of these relays are highly likely less than 100KiB/s each.

B. Large Performance Deviation

In this section, we will explore the service quality that these low-bandwidth relays can provide by performing an experiment to measure their performance. We customize a Tor client and deploy an exit relay (namely "KUITTC") running Tor v0.2.5.6-alpha in the campus network of KU. The customized client disables the default circuit establishment, and instead constructs two-hop circuits exiting at our KUITTC exit node. The first hop of a circuit is the relay that we want to measure, which is all active relays in the Tor network. The measurements are carried out by downloading a 1MB file through the tested circuits from an HTTP server, which is also located in the campus network. The client is implemented in Python and constructs circuits using Stem [19]. Since client, KUITTC and HTTP server all run on machines with very powerful CPUs and 10Gbps to the Internet, the only possible bottleneck in the experiment is the tested first-hop relays. Therefore, we consider the circuit performance as the performance of the tested relays. We measure all relays in the Tor network three times from December 23, 2014 to January 5, 2015 to calculate the average. The download timeout is set to 60 seconds to avoid long waiting.

Results. 6,053 out of 7,069 measured relays successfully finish the downloading task within 60 seconds. The remaining relays fail due to various reasons, including circuit construction failure, relay failure during downloading or download timeout. For the successful downloads, the average download time is 8.11 seconds with a standard deviation of 9.69 seconds and a median of 4.06 seconds. The significant standard deviation and the large difference between mean and median indicate that the capacity of relays deviates dramatically from each other.



(a) Used relays v.s. Idle relays

(b) fast relays v.s. slow relays in used relays

Fig. 2: Performance comparison of used relays and idle relays: download time measured in downloading 1MB files.

We further compare the performance of idle relays and used relays identified in the last experiment. From Figure 2a we can see that the performance of idle relays is much worse than that of used relays. Because 97.7% of idle relays are low-bandwidth relays, it is consistent with the expectation of the current Tor path selection scheme that aims to avoid slow routers. However, the current path selection does not completely exclude slow routers, the clients still have some chance to select them. Figure 2b compares the performance of fast relays ($\geq$ 100 KiB/s bandwidth) and slow relays ($<$100 KiB/s bandwidth) in the set of used relays. It is obvious that relays with less than 100KiB/s are much slower. Although they are selected in the experiment, the clients selecting them will have very poor user experience. This experiment answers the second question: the service quality of these low-bandwidth relays are very low, so interactive users should avoid them.

C. Discussion.

Based on the above experiment results, it seems that the current path selection algorithm is confronting a paradox: *poor performance* – selecting low-bandwidth relays indicates a higher chance to experience poor performance vs. *low utilization* – avoiding them means that many bandwidth resources are under-utilized. No existing solution including path selection optimization and limiting bulk traffic discussed previously can solve the paradox. However, the fact that under the current Tor design a large number of low-bandwidth relays are indeed idle, but they should not be used by interactive users for the purpose of performance shows potential to support bulk applications. If we can make use of these low-quality resources to transfer bulk traffic, the relay utilization will be increased and the traffic on high-bandwidth relays will be reduced, so that interactive users will also benefit from it.

IV. Multi-Path Tor Design

In this work, we propose a multi-path routing scheme, namely *m*Tor, to solve the performance paradox caused by bulk traffic. As illustrated in Figure 3, *m*Tor constructs an anonymous *tunnel* consisting of m *circuits* where m is a client-specific parameter. Different from the throttling schemes,

*m*Tor supports bulk traffic by distributing it to multiple low-bandwidth relays. This mitigates potential congestion on high-bandwidth relays and thus addresses the *low utilization* aspect of the paradox. On the other hand, while the capacity of each circuit is limited and dynamic over time, the proposed *m*Tor scheme can adaptively distribute bulk traffic onto circuits in proportion to their dynamic capacities and thus achieve an acceptable overall performance. *m*Tor is transparent to regular Tor relays employed in the first two hops. However, small modifications are needed at the edge components, i.e., exit relays and OPs of the clients with bulk traffic transfer needs, for associating multiple circuits to a client stream, adding sequence number to data cell, reordering out-of-sequence cells and scheduling cells across multiple circuits. Next, we elaborate the process of tunnel construction and data transmission.

A. Tunnel Construction

Relay selection. As shown in Figure 3, a *m*Tor tunnel consists of *m* circuits across different relays. To implement this design, small modifications are made to the current path selection scheme of Tor, mainly in the selection of middle relays. In particular, *m*Tor employs middle relays from a set of low-bandwidth relays (e.g., relays with less than 100KiB/s bandwidth) to avoid bulk data applications scrambling with interactive applications for the scarce fast routers. For entry and exit nodes, *m*Tor follows the current scheme to select routers in proportion to their weighted bandwidth. Circuits in a tunnel recruit different entry relays but share a same exit on which packets of bulk traffic applications are reassembled before reaching the destination. Note that employing multiple entry relays may increase the chance that a selected entry is controlled by an attacker and thus affect the anonymity provided by the system. Details will be discussed in Section VI.

Tunnel initialization and management. Atop circuit management of Tor, *m*Tor introduces additional tunnel management to initiate and group multiple circuits into a tunnel for bulk traffic transmission. After relay selection, the client OP constructs a first circuit and sends a *relay_multipath* cell to the selected exit relay to request a multipath connection. In response, the exit relay generates a unique 32-bit tunnel identifier (*TID*) and incorporates it in the replied *relay_multipath_ack*. To join the tunnel, OP sends a *relay_join* cell, which contains *TID*, through each established circuit so that the exit can link all circuits with the same *TID* together to form a multipath tunnel. The exit acknowledges a successful joining by sending back a *relay_joined* cell. Once the OP receives $m-1$ acknowledgements, a multipath tunnel is successfully constructed. Note that all the cells are layered encrypted so only OP and the exit at two ends see *TID*. This prevents the entry and middle nodes from linking *TID* with a specific client.

*m*Tor manages multipath tunnels dynamically according to the congestion status of member circuits over time. If OP detects that the transmission on a member circuit becomes very slow, it will construct a new circuit to replace the slow one. We will discuss how to define "slow" in Section IV-B. The slow-circuit-closing scheme provides OP the ability of responding real-time network dynamics, which prevents a slow circuit from becoming a bottleneck of the entire tunnel. Besides tearing down slow circuits, an *m*Tor circuit is replaced by a new one when it gets "dirty" - with a lifetime exceeding 10 minutes and no streams on it, similar as in Tor circuit management. This also makes periodical tunnel reconstruction unnecessary, because member circuits of a tunnel are frequently updated.

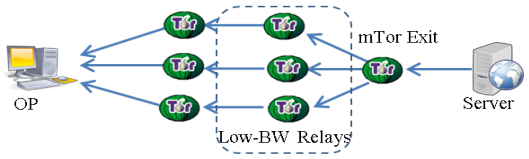


Fig. 3: mTor architecture.

B. Data Transmission

When a bulk application stream arrives at OP via SOCKS, OP spawns the client stream (denoted as the *parent* stream) to *m* subflows and appends them to the tunnel by associating each subflow with a member circuit. Each subflow has its own stream window and inherits a common stream ID from the parent stream. Next, OP sends a *relay_begin* cell through a randomly selected member circuit to inform the exit about the stream ID and the destination, and then starts data transmission.

Scheduling and data cell allocation. Conceptually, data cells can be forwarded through any member circuit in the tunnel. However, if the number of allocated cells on a particular circuit exceeds its capacity, it will become congested. Since the overall performance of a tunnel is bounded by the slowest circuit, two endpoints of a tunnel need to cooperate to schedule cells across multiple circuits based on the capacities of individual circuits. A straightforward approach for cell allocation is to probe the capacity of each circuit after it is initiated and schedule traffic in proportion to the probed capacity. However, the method is unrealistic and less accurate. First, it will introduce a large amount of extra traffic to the Tor network. Moreover, the capacity of each circuit may change dramatically after the probing. Therefore, the static scheduling scheme cannot adapt to network dynamics which makes the allocation less useful. In [20], Alsabah et al. presented an opportunistic probing approach to estimate the round-trip-time (RTT) of a circuit based on Tor's circuit-level congestion control scheme, where RTT is measured as the difference between the time of sending out every 100^{th} cell and the time of receiving a circuit *SENDME* flag. The number of cells allocated to a circuit is reversely proportional to the measured RTT. The RTT-based approach is reactive to network dynamics, but it is still not very accurate because the congestion feedbacks are received infrequently (every 100 cells) [21]. Besides, cells may spend several hundred milliseconds in the circuit queue before being transmitted over a TCP connection [7], after which the circuit's capacity may change non-negligibly. Based on this consideration, we argue that RTT-based approach may not be a good choice in cross-layer scheduling, which is also recognized in multipath TCP design [22]. In *m*Tor, we adopt a "pulling" scheduling – instead of pushing data cells to circuits by a scheduler, we let each subflow actively pull data from a shared send buffer whenever its stream window becomes nonempty. Initially, each subflow has a stream window of 500 cells. As described in Section II, the stream window decreases

by one when sending a cell out and increases by 50 when receiving a stream-level *SENDME* notification. Consequently, a subflow stops sending cells when its stream window size drops to zero and resumes when it receives a *SENDME*. It is obvious that when the circuit to which a subflow is appended becomes congested, cells will be moving much slowly towards the receiver resulting in delayed stream-level *SENDME*s and long waiting at the sender end. Whereas, subflows on fast circuits will send out data cells steadily. In this way, the "pulling" scheduling is subflow self-adaptive without accurate explicit circuit RTT measurements. When multiple subflows have a nonzero stream window, we adopt a FIFO (first-in-first-out) queue to schedule them.

Slow circuits detection. Another challenging issue in *m*Tor design is the detection of slow circuits. Since *m*Tor recruits low-bandwidth routers, its circuits are more likely to be congested. To avoid a congested circuit becoming the bottleneck of the entire tunnel, OP will construct new circuits to replace the slow ones. In this work, we adopt a distance-based outlier detection approach to determine whether a circuit is congested based on a sliding window of 50 cells. In particular, we measure the time of receiving every 50 cells and find the lower and upper quartiles (Q_1 and Q_3) of ten most recent records to calculate the interquartile range (IQR) where $IQR = Q_3 - Q_1$. If a new measurement falls out of the range of $(0, Q_3 + 1.5IQD]$, it is considered as an outlier indicating the circuit is experiencing a congestion. To increase detection reliability, OP considers a circuit as congested if at least three consecutive outliers occur. Once detected, OP and the exit will collaborate to tear down the congested circuit and replace it with a new one. First, OP informs the exit to drop the congested circuit via a *relay_drop* message with the number of cells already received on the circuit (denoted as n_r). Next, OP initiates a new circuit and requests to join the existing tunnel by sending a *relay_join* message with the same *TID*. After receiving *relay_drop*, the exit immediately stops sending cells on this circuit and compares n_r with the number of cells already sent, denoted as n_s. If n_r equals to n_s, a *relay_dropped* message will be sent to inform OP to tear down the circuit. Otherwise, a *relay_stopped* message will be sent to inform OP to keep receiving the remaining $n_s - n_r$ cells from the circuit before tearing it down.

Data re-ordering. Combining the self-adaptive "pulling" scheduling and active congestion detection schemes, *m*Tor is able to adapt to network dynamics, which potentially prevents slow circuits from degrading the overall performance of multipath tunnels. However, due to dynamic scheduling, data cells may arrive at the receiver out of order. To solve this issue, we have to modify the format of Tor data cell to incorporate a 32-bit sequence number in the multipath data packets. As shown in Figure 4, the first four bytes of data payload is reserved for this purpose. Moreover, we add a new *relay_subdata* command to indicate a data cell is multipath data. When OP receives a multipath data cell from a subflow, it first checks if the sequence number is what it expects. An expected cell is immediately forwarded to the application stream, while an out-of-order cell is stored in a buffer and ordered according to its sequence numbers.

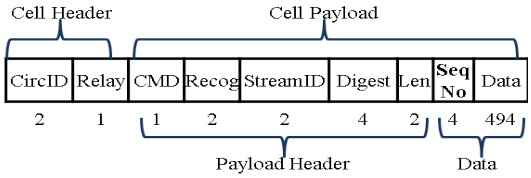


Fig. 4: Cell format.

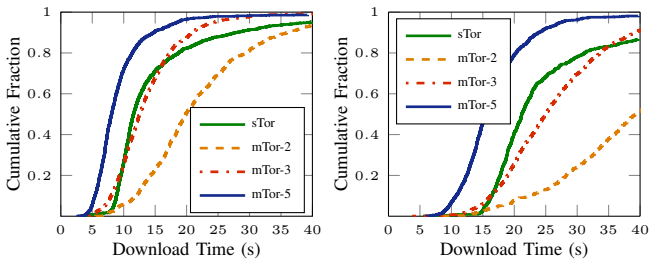
V. Performance Measurements

To explore the performance improvements introduced by *m*Tor, we perform experiments on the live Tor network and compare the average download time of the standard single-path Tor (denoted as sTor) with the one of *m*Tor. We also conduct simulations on the Shadow simulator [1] to study the impacts of *m*Tor on interactive traffic.

A. Measurements on Live Tor

Setup. In the live Tor experiments, a client is asked to download large documents from a web server over HTTP to simulate bulk data traffic. Since *m*Tor requires modifications on the exit node, we deploy our own exit relay (namely "KUITTC") in the Tor network and recruit it as the exit in all the experiments. Both "KUITTC" exit and the web server are located in the campus network of KU, so the transmission latency of the last step is very small. For sTor, the client machine running Tor v0.2.5.6-alpha follows the current Tor path selection algorithm to select the entry and middle relays from the Tor network and constructs a three-relay circuit. For multipath routing, the client OP follows the proposed *m*Tor design with *m* varying from 2 to 3 and 5.

Results. Since the design goal of *m*Tor is to support bulk data applications without affecting interactive traffic, download time is considered a more important metric than latency.Therefore, we measure the download time to compare the performance of sTor and *m*Tor. Figure 5 shows the cumulative distribution of the download time for 5MB and 10MB files through sTor and *m*Tor. From the figure, we see that *m*Tor-2 with a tunnel of two circuits performs worst. This indicates the combined capacity of a small number of slow paths may not be enough to compensate for the low capacity of each circuit. However, the performance of *m*Tor increases fast along with the increase of m. When downloading 5MB documents, *m*Tor-3 can achieve a comparable performance as sTor does and the CDF of *m*Tor-3 has a shorter tail regardless of file size. We further measure the standard deviation as 19.76s and 10.34s for sTor and *m*Tor respectively when downloading 5MB file, and 23.98s and 12.33s respectively when downloading 10MB file. The much smaller standard deviation of *m*Tor indicates desirable small performance deviation. This is because *m*Tor is more sensitive and adaptive to circuit-level congestions, while sTor is less responsive to the single congested path. We also see that in both settings, *m*Tor-5 outperforms sTor, indicating that even with low-bandwidth relays, a bulk data application can achieve a comparable performance with a reasonable large m. More importantly, as *m*Tor becomes more attractive to bulk data applications, it will shift the load from high-bandwidth routers to low-bandwidth ones and thus potentially benefit



(a) 5MiB download time (b) 10MiB download time

Fig. 5: Comparison of measured time of downloading 5MiB and 10MiB files through sTor, mTor-2, mTor-3 and mTor-5.

interactive traffic. Next, we will investigate the impact of *m*Tor on reducing the load of high-bandwidth relays.

B. Measurements on Shadow

Setup. The Shadow simulator [1] is a discrete event simulator implementing Tor algorithms over a simulated network topology. We plug *m*Tor into Shadow to build a private Tor network and compare the performance of web traffic and the load on high-bandwidth relays using sTor and *m*Tor. In particular, we build a small network with 50 Tor relays, 200 HTTP servers, 990 web clients, and 110 bulk clients, where the ratio of the number of relays to the number of clients is proportional to the real distribution in the Tor network, the ratio of different types of relays is sampled from the Tor network statistics in February 2015, and the ratio of two types of clients follows client distribution suggested in [6]. Simulating web browsing behavior, web clients download a 320K file, as of an average webpage size, from a randomly selected HTTP server and wait for between 1 and 60 seconds before starting the next download. The bulk clients repeatedly download a 5M file with no stop between any two downloads. For sTor, all clients and relays adopt the default Tor setting, while for *m*Tor all bulk clients and exit relays operate in the multipath mode.

Results. Since web clients are delay-sensitive, we consider two metrics, *time to first byte* (TTFB) and *download time*, in evaluating the performance for web traffic. We first investigate the impact of bulk traffic to interactive traffic. Figures 6a-6h show the performance of web clients with bulk traffic load varying from *light* (with 110 bulk clients) to *medium* (with 310 bulk clients) and *heavy* (with 510 bulk clients). *m*Tor (with $m = 3$ and 5) demonstrates smaller TTFB and shorter download time than sTor under three types of loads, indicating improved network responsiveness and overall download time. This result is consistent with our expectation that *m*Tor can reduce the load on high-bandwidth relays, which are frequently recruited by web clients. This is because by restricting the selection of middle nodes among only the low-bandwidth relays, *m*Tor preserves more resource on high-bandwidth relays for web traffic and mitigates potential congestion.

To demonstrate that *m*Tor reduces the contention of bulk clients for high-bandwidth relays with web clients, we further compare the bandwidth consumption ratio of bulk traffic on fast routers and slow routers in both sTor and *m*Tor. In particular, we record the number of bytes that each relay reads and writes per second on all circuits to calculate the overall bandwidth consumed by each client on each relay along a circuit. As shown in Table I, under medium load, the bandwidth consumed by bulk traffic on all fast routers decreases from 19.86% of the overall bandwidth in sTor to 14.69% in *m*Tor-3, while the ratio on slow routers increases from 17.33% in sTor to 95.16% in *m*Tor-3. In the case of heavy load, load shifting from fast routers to slow routers is more obvious, dropping from 34.96% to 15.30%.

	Medium load		Heavy load	
	sTor	*m*Tor-3	sTor	*m*Tor-3
Fast router ($\geq 100KB/s$)	19.86%	14.69%	34.96%	15.30%
Slow router ($< 100KB/s$)	17.33%	95.16%	30.32%	95.74%

TABLE I: Comparison of bandwidth consumed by bulk clients on fast routers and slow routers for sTor and *m*Tor under medium load and heavy load.

Finally, we compare the performance of bulk traffic in sTor, *m*Tor-3 and *m*Tor-5 under different network loads. As shown in Figure 6c, 6f and 6i, sTor outperforms *m*Tor-3 and *m*Tor-5, which slightly deviates from the results obtained from previous live experiments. This indicates that m should be larger than 5 for an approximate performance in sTor. In fact, this inconsistency implies that the real capacity of slow routers in the live Tor network is greatly underestimated. For instance, the authority directory often assigns a lower bandwidth consensus weight than its real capacity to a newly joined relay in the early time of its lifecycle [23]. On the contrary, the advertised bandwidth is the exact capacity of the simulated relays. Another possible reason is the comparably small set of low-bandwidth routers in the simulated network. When a large number of bulk traffic circuits go through a small number of low-bandwidth routers, they are easily overloaded. This is not the case in the live Tor network, in which over 80% relays are low-bandwidth routers.

VI. Anonymity Analysis

Grouping multiple paths in a tunnel may result in degraded anonymity, especially when under the traffic correlation attacks [24]. In this section, we will study the impact of the proposed multipath Tor on the anonymity of a 3-hop tunnel and discuss the potential vulnerabilities introduced by *m*Tor.

A. Adversary Model

It is widely acknowledged that Tor provides good anonymity if at least one of the three relays in a Tor circuit is honest. However, the anonymity degrades due to traffic correlation if the first and last relays are controlled by an adversity. Known as the traffic correlation attacks [17], [24], [25], the attacker is assumed to be able to monitor two endpoints from which clients' traffic enters and leaves the Tor network and perform traffic analysis to link the sender and receiver regardless of the intermediate relay. Therefore, we assume there exists a *relay adversary* who controls a small number of geographically distributed Tor relays (e.g., through a botnet). We further assume the relay adversary can add and alter the traffic over its controlled relays.

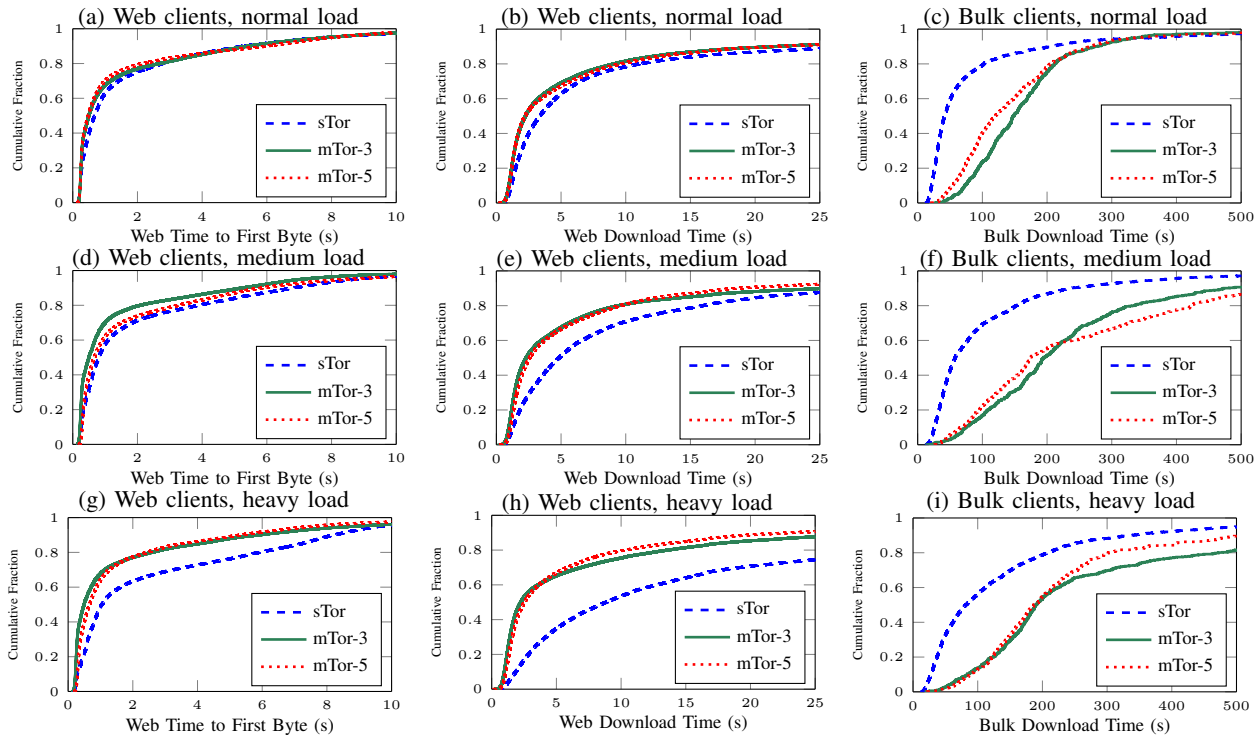


Fig. 6: Comparison of client performance for sTor (the standard single-path Tor), mTor-3 and mTor-5 under fixed web load (990 web clients) and different bulk load changing from light load (110 bulk clients for 6a-6c), to medium load (310 bulk clients for 6d-6f), and to heavy load (510 bulk clients for 6g-6i).

B. Correlation Risk

The concept of entry guards is introduced in the current Tor implementation to mitigate the threat of traffic correlation attacks. Entry guards are a set of reliable, high-bandwidth relays trusted by the client to play as the entry node. Instead of picking a new entry relay for each circuit, a client maintains a set of entry nodes for a period of time and randomly selects an entry relay from this set for each circuit. By default, an entry guard set has a size of 3 and expires after 30 to 60 days.

If no relay in the guard set is controlled by the adversary, the client is completely secure, otherwise, there exists a small chance of being compromised (i.e., correlated). We can calculate the probability of a successful compromise as follows:

$$P_{compromised} = \sum_{a=0}^{g} P(a) \cdot P(compromised|a)$$

where g represents the size of guard set ($g = 3$ by default) and a denotes the number of adversary relays in the guard set. $P(a)$ is the probability that the guard set contains a adversary relays, and $P(compromised|a)$ is a conditional probability of being compromised given a adversary relays in the guard set. Given the ratio of the number of compromised relays over the number of overall relays, a follows the binomial distribution:

$$P(a) = \binom{g}{a} \cdot f_{gbw}^{a} \cdot (1 - f_{gbw})^{g-a}$$

where f_{gbw} and f_{xbw} represent the fraction of bandwidth of malicious guards and exits with respect to the total bandwidth of guards and exits in Tor, respectively. Therefore, the conditional probability $P(compromise|a)$ can be computed as:

$$P(compromised|a) = f_{xbw} \cdot \left[1 - \frac{\binom{g-a}{m}}{\binom{g}{m}}\right]$$

where m is the number of circuits in a tunnel and $\binom{g-a}{m} = 0$ if $g - a < m$. In sTor, m equals to 1. However, in *m*Tor, if $m \geq g$, the formula can be rewritten as $P(compromised|a) = f_{xbw} \cdot [1 - \binom{g-a}{g}]$, because m circuits will use up all g entry relays in the guard set. As a result, there is a high risk of recruiting a malicious relay as the entry. If any of m circuits chooses both the malicious entry and exit, the client is considered compromised.

Now we quantify the correlation risk of *m*Tor clients. First, we compute the risk under the general literature setting [20], that is, less than 20% bandwidth are controlled by an adversary. Figure 7 compares the probabilities of compromise with different m under fixed malicious exit bandwidth and varying malicious guard bandwidth. *m*Tor increases the probability from 4% of sTor to 13.45% of *m*Tor-5.

In the above analysis, the adversary who controls 20% of guards and 20% of exits may be more powerful than the reality, so we then will discuss how vulnerable *m*Tor is under a realistic adversary. To determine the amount of bandwidth controlled by a relay adversary, in Table II we list the top five families with the largest bandwidth on March 20, 2015. Referring to Table II, we assume that a realistic relay adversary controls totally 240MiB/s bandwidth. To maximize

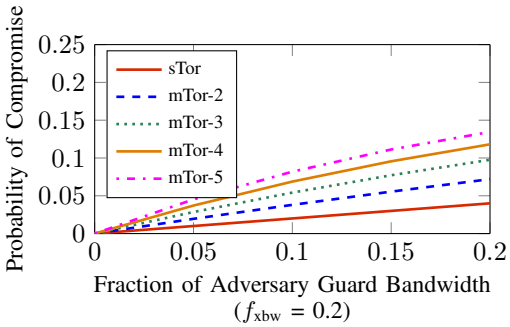


Fig. 7: Compromise probability for mTor with different *m*.

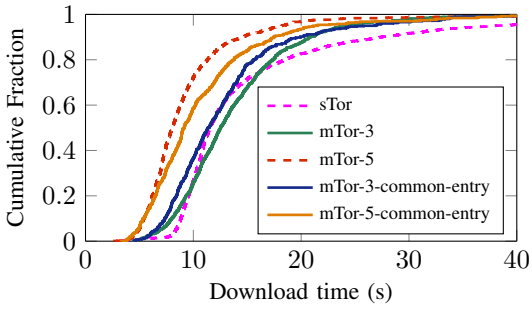


Fig. 8: Comparison of measured time of downloading 5MiB files through sTor, mTor-3, mTor-5, mTor-3-common-entry and mTor-5-common-entry.

the probability of compromising a client, the adversary must reasonably allocate his limited bandwidth to entry guards and exit nodes. We follow the finding in [25] that 5:1 entry:exit ratio maximizes the chance that the adversary successfully compromises at least one circuit.

Instead of directly using advertised bandwidth, the path selection of Tor is based on consensus weight of each relay, which is computed by directory authorities via measuring the relay bandwidth and then applying PID control feedback. Because computing the consensus weight is a complicated process, and considering the fact that the consensus weight of each relay is related to its advertised bandwidth, for the sake of simplicity, the consensus weight of the adversary can be referred to the family members of "kalach" in Table II, which has approximate total advertised bandwidth. The total consensus weight of kalach's family is 560,210, so 466,841.7 is allocated to guards while 93,368.3 is allocated to exits according to 5:1 ratio. We also investigate the total consensus weight of relays in Tor network, which are guard-only (17,761,271), exit-only (983,520) and guard&exit (9,466,357) on March 20, 2015, respectively. After adjustment by bandwidth weights (e.g. Wg and We) in the consensus on March 20, 2015, f_{gbw} and f_{xbw} are 2.63% and 4.47%, respectively. Compared to the adversary setting in the literature, a realistic adversary controls much less bandwidth resource. Table III shows the probability of the adversary compromising a *m*Tor client with the increase of tunnel width *m*. When *m* equals 1, it represents Tor. From the result we can see that confronted with a realistic relay adversary the deanonymization risk of *m*Tor is very low.

Rank	Bandwidth (MiB/s)	Rep. Family Mem.
1	526.4	theoden
2	263.4	NCC75633
3	243.0	kalach
4	192.4	AccessNow011
5	159.3	torpidsCZgtt

TABLE II: Top 5 relay families with the highest advertised bandwidth which is the lesser of observed bandwidth and configured bandwidth.

Obviously, the more circuits are used by a *m*Tor client, the higher risk of being compromised it takes. As we have seen in Section V-A, to achieve comparable or better performance, a m-Tor user should set *m* to 3 or greater, which means a *m*Tor user will have three times higher risk of being compromised than sTor as Table III shows.

To avoid sacrificing any anonymity, a *m*Tor client can force *m* circuits to share a common entry guard so that *m*Tor can achieve the same anonymity level like sTor. Figure 8 compares the performance of downloading 5MiB files between *m*Tor with the common entry guard, original *m*Tor and sTor. From the Figure 8, we can see that the performance of *m*Tor-5 with common entry guard is slightly worse than original *m*Tor-5, because multiple circuits have to share the bandwidth of a single guard relay, which reduces the available bandwidth for each circuit. Overall, we can see that even if a *m*Tor client uses a single entry guard for his all circuits in the tunnel, he can still achieve comparable or even better performance than he uses sTor.

Therefore, clients can adjust *m* and the entry guard to balance the tradeoff between performance and security. For a bulk client requiring high anonymity, he can choose small *m* and a common entry guard for his tunnel to achieve acceptable performance without sacrificing anonymity, while for a client preferring performance more, he can adopt the default setting and choose a larger *m*.

m	1(Tor)	2	3	4	5
P_{com}	0.12%	0.23%	0.34%	0.45%	0.56%

TABLE III: The probability of a *m*Tor client being compromised by a relay adversary.

VII. Related Work

Plenty of work has been proposed to improve Tor's performance for interactive users. LASTor [13] takes into account the inferred geographic locations of relays in path selection to reduce latency. Snader and Borisov [4] propose a tunable path selection algorithm according to relays' relative ranking which is assigned based on the median of the peak bandwidth that all other relays recently have seen. These approaches prefer relays with high confidence to provide good services based on their performance history, however, they do not consider the current load of relays, which may affect a relay's real performance. The new implementation of Tor path selection considers circuit construction time as the current load indicator and drops circuits with long construction time, but it cannot deal with congestion which occurs after the circuit establishment. Another congestion-aware path selection approach [5] selects relays with good congestion history and actively detects

congestion during data transfer. Once congestion is detected, it switches to another idle circuit instead.

Tang and Goldberg propose a new circuit scheduler which prioritizes interactive traffic over bulk traffic based on the exponentially weighted moving average (EWMA) of relayed cells [7]. However, experiments on simulator show that EWMA-based scheduler highly depends on network and load, so it is not clear if performance will always improve [1]. Different from scheduling scheme, more aggressive approaches have been proposed to throttle bulk traffic. Jansen et al. proposes EWMA-based adaptive throttling algorithms that select the busiest connections on entry node to throttle and dynamically adjust the throttle rate of each connection [8].

Another multipath routing scheme "Conflux" [20] over Tor is proposed to improve the performance for bridge and video streaming users, which first implements the multipath scheme in Tor and provides a thorough anonymity analysis. However, Conflux still adopts the default path selection scheme so that a bulk client have multiple circuits contending for high-bandwidth relays, which is unfair to regular interactive users. Our work, despite the adoption of the same multipath idea like "Conflux", has different insights to relay utilization and new implementations for path selection, tunnel management and circuit scheduling, which utilizes unused bandwidth to support bulk traffic with better fairness.

VIII. Conclusion

In this paper, we propose a multipath Tor (*m*Tor) routing scheme to enable Tor to support bandwidth-intensive applications without hurting the interactive clients. We explore the difficulty in utilizing low-bandwidth routers in the current Tor architecture, that is, if web clients use them, they will experience very poor performance; otherwise, a large amount of bandwidth resource would be under-utilized. *m*Tor solves this problem and serves bulk clients using low-bandwidth relays to reduce the effects of bulk traffic on interactive traffic. Meanwhile it utilizes multipath routing scheme to make up the low capacity of slow relays and avoids congestion on a single circuit by dynamically allocating traffic across multiple circuits according to their capacities. In the live experiments and Shadow simulator, we show that *m*Tor can achieve desirable performance for bulk clients and also benefits the web clients by reducing the load on high-bandwidth relays. Anonymity analysis shows that *m*Tor slightly increases the chance that a bulk client is compromised, but users can adjust the tradeoff between performance and anonymity by changing *m* and limiting the number of distinct entry guards in the tunnel.

Acknowledgments

This work was partially supported by the National Science Foundation under Award EPS0903806, KU General Research Fund under Award GRF2301075, and KU Research Investment Council Strategic Initiative Grant under Award INS0073037.

References

[1] R. Jansen and N. Hopper, "Shadow: Running Tor in a Box for Accurate and Efficient Experimentation," in *Proceedings of the Network and Distributed System Security Symposium - NDSS'12*, February 2012.

[2] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[3] TorProject. Estimated Number of Clients in the Tor Network. https://metrics.torproject.org/clients-data.html.

[4] R. Snader and N. Borisov, "A tune-up for Tor: Improving security and performance in the Tor network," in *Proceedings of the Network and Distributed Security Symposium - NDSS '08*, February 2008.

[5] T. Wang, K. Bauer, C. Forero, and I. Goldberg, "Congestion-aware Path Selection for Tor," in *Proceedings of Financial Cryptography and Data Security*, 2012.

[6] D. McCoy, K. Bauer, D. Grunwald, T. Kohno, and D. Sicker, "Shining light in dark places: Understanding the Tor network," in *Proc. of the 8th International Symposium on Privacy Enhancing Technologies*, 2008.

[7] C. Tang and I. Goldberg, "An improved algorithm for Tor circuit scheduling," in *Proceedings of the 2010 ACM Conference on Computer and Communications Security (CCS 2010)*, October 2010.

[8] R. Jansen, P. Syverson, and N. Hopper, "Throttling Tor Bandwidth Parasites," in *Proceedings of the 21st USENIX Security Symposium*, August 2012.

[9] R. Jansen, N. Hopper, and Y. Kim, "Recruiting new Tor relays with BRAIDS," in *Proceedings of the 2010 ACM Conference on Computer and Communications Security (CCS 2010)*, October 2010.

[10] T.-W. J. Ngan, R. Dingledine, and D. S. Wallach, "Building Incentives into Tor," in *Proceedings of Financial Cryptography*, January 2010.

[11] W. B. Moore, C. Wacek, and M. Sherr, "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 (ACSAC'11)*, December 2011.

[12] R. Jansen, A. Johnson, and P. Syverson, "LIRA: Lightweight Incentivized Routing for Anonymity," in *Proceedings of the Network and Distributed System Security Symposium - NDSS'13*, February 2013.

[13] M. Akhoondi, C. Yu, and H. V. Madhyastha, "LASTor: A Low-Latency AS-Aware Tor Client," in *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, May 2012.

[14] M. AlSabah, K. Bauer, and I. Goldberg, "Enhancing tor's performance using real-time traffic classification," in *Proceedings of the 19th ACM conference on Computer and Communications Security*, October 2012.

[15] Cisco. VNI Forecast Highlights. http://www.cisco.com/web/solutions/sp/vni/vni_forecast_highlights/index.html.

[16] M. Kende. Global Internet Report. http://www.internetsociety.org/doc/global-internet-report.

[17] L. Øverlier and P. Syverson, "Locating hidden servers," in *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, May 2006.

[18] M. Wright, M. Adler, B. N. Levine, and C. Shields, "An analysis of the degradation of anonymous protocols," in *Proceedings of the Network and Distributed Security Symposium - NDSS '02*, February 2002.

[19] D. Johnson, "Stem: a Python controller library for Tor," https://stem.torproject.org.

[20] M. Alsabah, K. Bauer, T. Elahi, and I. Goldberg, "The path less travelled: Overcoming tor's bottlenecks with traffic splitting," in *Proc. of the 13th Privacy Enhancing Technologies Symposium*, July 2013.

[21] M. AlSabah, K. Bauer, I. Goldberg, D. Grunwald, D. McCoy, S. Savage, and G. Voelker, "Defenestrator: Throwing out windows in tor," in *Proc. of the 11th Privacy Enhancing Technologies Symposium*, July 2011.

[22] S. Barre, C. Paasch, and O. Bonaventure, "Multipath tcp: from theory to practice," in *NETWORKING 2011*. Springer, 2011, pp. 444–457.

[23] R. Dingledine. The Lifecycle of a New Relay. https://blog.torproject.org/blog/lifecycle-of-a-new-relay.

[24] S. J. Murdoch and G. Danezis, "Low-cost traffic analysis of Tor," in *Proc. of the 2005 IEEE Symposium on Security and Privacy*, May 2005.

[25] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson, "Users get routed: Traffic correlation on tor by realistic adversaries," in *Proceedings of the 20th ACM conference on Computer and Communications Security*, 2013.