TorScan: Tracing Long-lived Connections and Differential Scanning Attacks

Alex Biryukov, Ivan Pustogarov, Ralf-Philipp Weinmann

University of Luxembourg

Abstract. Tor is a widely used anonymity network providing low-latency communication capabilities. Around 400,000 users per day use Tor to route TCP traffic through a sequence of relays; three hops are selected from a pool of currently almost 3000 volunteer-operated Tor relays to comprise a route through the network for a limited time. In comparison to single-hop proxies, forwarding TCP streams through multiple relays increases the anonymity of the users significantly: each hop along the route only knows its successor and predecessor. The anonymity provided by Tor heavily relies on the hardness of linking a user's entry and exit nodes. If an attacker gains access to the topological information about the Tor network instead of having to consider the network as a fully connected graph, this anonymity may be reduced. In fact, we have found ways to probe the connectivity of a Tor relay. We demonstrate how the resulting leakage of the Tor network topology can be used and present attacks to trace back a user from an exit relay to a small set of potential entry nodes.

1 Introduction

Anonymity clearly was not a concern when the Internet Protocol was designed. Hence it comes as no surprise that internet communications are traceable. Today, the consequences of linking your traffic profile to your persona vary: it ranges from ISPs selling your aggregated web browsing history to marketers in democratic countries to being imprisoned for criticizing the government online in countries with repressive regimes. For many people, the first approach to hiding their identity is a public proxy server. This however is no panacea: the owner of the proxy can be forced to reveal any logs potentially stored – or even worse, the server may turn out to be a honeypot of the organization you are trying to hide from.

A better solution is to forward traffic through a chain of network nodes, so-called *relays*. This idea was developed by David Chaum in 1981 in his seminal paper [3] on mix networks. A *mix* is a fundamental building block in an anonymity network that hides the input/output relationship of the messages that pass through it. A number of implementations of mix networks have been built, notably Mixmaster and Mixminion for making email communication untraceable; moreover mix networks are used in a number of e-voting protocols. To guarantee good anonymity, a mix network usually delays messages and inserts chaff. This however is not acceptable for low-latency communications.

In 1996, Goldschlag, Reed and Syverson [5] presented *Onion Routing*, a design limiting traffic analysis on low-latency communication that was inspired by Chaum's mix networks. Tor is the refined successor of the original Onion Routing Project. The Tor network is a low-latency anonymity network which at the time of writing comprised of 2500-3000 routers with an estimated number of daily users (unique IPs) exceeding 400,000. Tor tries hard to achieve low traffic latency to provide a good user experience, thus sacrificing some anonymity for performance. To keep latency low and network throughput high, Tor relays do not delay incoming messages and do not use padding.

One way to undermine the anonymity of a Tor user is to reveal the pair of the corresponding entry and exit node; this is supposed to be hard. Once the correspondence between the entry and exit nodes is known, the anonymity of the observed connection is reduced to the case of two known sequentially connected proxies, or to the case of a single proxy if the attacker controls the exit node. Though this will not allow us to immediately determine the actual originator of the connection, this is already a significant information leak because triplets of guard nodes can serve as unique user identifiers within the Tor network, and also because knowing the entry node tells the attacker where to target next. Namely, other attack may be launched to compromise the entry node, or the entry node's operator/ISP could be presented with legal demands to reveal the network logs. Given that the exit node is known, the probability of correctly guessing the entry node is $\frac{1}{n}$, where n is the number of guards in the Tor network. For an adversary with less visibility than a global passive adversary and a fully connected network, increasing this probability is far from straightforward. Still in reality, not all entry and exit nodes are connected via three hop paths (which is default for Tor) at a given point of time. This observation can become the basis of several novel attacks on Tor, as will be shown in the paper: The main contributions of this paper are:

(i) We present two ways to reveal the connectivity of nodes in the Tor network: one using canonical connections which are a part of the Tor specification; the other is a more generic technique, namely a timing attack on the connection establishment between two relays.
(ii) We present novel attacks which are based on the *connectivity scanning* approach. The first attack allows to identify the guard node which was used in a circuit carrying a long-lived connection – such as an SSH session or a large file download. The second attack, which we have chosen to call *differential scan attack*, uses recurrent connections to reveal all guard nodes of a user.
(iii) We give some guidance on countermeasures that can be implemented to make the Tor network more resilient to leakage of topology information.

The rest of the paper is organized as follows: in the next section, we summarize aspects of the Tor specification which are relevant for the connectivity scanning techniques and for the description of our attacks. Thereafter we give a short overview of previous attacks on Tor. We describe our techniques for revealing the connectivity of Tor relays in Section 3. In Section 4.1, we describe our attack on long-lived streams. The differential scan attack is described in Section 4.2. An analysis of the attacks is performed in Section 5. We discuss the potential countermeasures in Section 6 and conclude in Section 7.

2 Background

Tor is a popular volunteer-based overlay network used to conceal a user's location or usage from adversaries conducting network surveillance or traffic analysis. Using Tor makes it more difficult to trace Internet activity for TCP applications. To connect to a server through Tor, a client first chooses a path (i.e. a sequence of the Tor relays: *guard*, *middle*, and *exit*) which will then carry data back and forth between the client and the server. To choose a path, the client downloads the list of available Tor routers from a *directory server*. Each Tor router in the list is uniquely identified by the SHA-1 message digest of its RSA public key. To prevent sampled profiling attacks each user has a fixed triplet of guard nodes which does not change for approximately one month. Each time the user needs to choose a guard node, he chooses it uniformly from this triplet.

After the sequence of relays is chosen, the client starts to build a circuit, one hop at a time. First, the user sets up a TLS connection with the guard node and negotiates a Diffie-Hellman (DH) key using `COMMAND_CREATE` and `COMMAND_CREATED` cells[1]. This creates a one-hop circuit. The client extends the circuit to the middle node through the guard node: he sends a `RELAY COMMAND_EXTEND` cell to the guard in which he specifies the address, the digest of the middle router, and the first step of DH key exchange encrypted by the middle node's public key. Once the guard node receives the cell, it establishes a TLS connection with the middle node and sends it the encrypted portion of the DH handshake. The middle node decrypts it and replies with the second step of DH exchange which is forwarded to the client within a `RELAY COMMAND_EXTENDED` cell. In this way the second hop of the circuit is established.

If during the circuit construction process the middle node rejects the connection, the guard node sends a `COMMAND_DESTROY` cell, specifying the error code, so that the client is forced to choose another sequence of relay nodes and try to construct a new circuit. If the circuit is extended successfully up to the middle node, the rest of the circuit is established in the same way. After the circuit has been built, the client can start transmitting and receiving data over this circuit. All TCP connections of the user's application are translated into Tor streams which are multiplexed over the circuit. Using the initially chosen circuit for a long time makes profiling attacks easier: the longer the duration of the circuit, the more time the attacker has to reveal it. For this reason, circuits older than 10 minutes are not allowed to carry new streams (for new streams a new circuit should be constructed.) After 10 minutes a circuit dies unless it carries a long-lived stream. In the latter case, the lifetime of the circuit equals the lifetime of

[1] Tor protocol messages are called "cells".

the long-lived stream. In other words, a circuit is not destroyed until at least one stream is attached to it. In a similar way, a TLS connection between two Tor relays is not closed if it carries at least one circuit. A TLS connection without circuits between two Tor routers lives for three minutes. There is one exception to the rule. A circuit which has never carried a stream (a *clean* circuit[2]) lives for 1 hour.

When a pair of Tor routers or a Tor router and a client have several circuits between them, they try to tunnel them over a single TLS connection. In Figure 1 communication between two Tor routers is shown. The routers use a single TLS connection (which is also called Onion Routing connection) which carries a number of circuits, two in this picture (which may belong to different end users). Multiple streams of one user may be multiplexed over a single circuit.

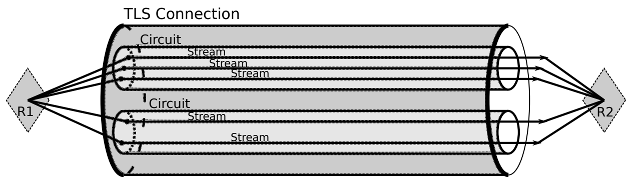

Fig. 1. Circuits and streams multiplexing

2.1 Related papers

Many different attacks on low-latency anonymity networks in general and on Tor network in particular are described in the literature. The most successful attacks can be categorized into passive traffic analysis attacks, active traffic analysis attacks, and attacks based on information leakage from specific applications. Passive traffic analysis attacks assume that an attacker passively observes a number of connections in an anonymity network and tries to correlate these connections either between themselves [4, 10, 1, 6, 2, 14] or with a predefined traffic pattern [9]. Active traffic analysis attacks assume that an attacker can inject traffic or delay traffic passing through the Tor network, thus modifying traffic and/or timing patterns of a targeted flow [8, 13, 12, 11]. Application specific attacks use the fact that applications may establish TCP connections directly (including connections to malicious servers) ignoring Tor and may establish UDP connections which are not supported by Tor [7]. Also, some applications may leak IP addresses in protocol messages.

The attacks presented in this paper do not require monitoring nor sending significant traffic at all (only a limited number of Tor protocol management cells) which makes these attacks relatively cheap. They also do not require the

[2] Once a new stream is attached to the circuit, it is marked as “dirty”

attacker to have the global view of the network which is needed by a number of passive traffic analysis attacks. In addition, the attacks presented in this paper are orthogonal to the previous attacks and thus can be used to improve some existing attacks making them more practical by reducing the traffic costs, or the number of monitored nodes (for ex. Murdochs attack [8]). Finally, the attacks presented here do not rely on the details of a particular user application or protocol.

3 Revealing Tor connectivity dynamics

Consider an attacker who wants to link the exit and the guard node of a circuit and thus decrease the anonymity of the user. Given the Tor network connectivity information, she can determine possible 3-hop paths from the exit node to the set of guard nodes and eliminate those which are impossible, thus already decreasing the claimed anonymity of Tor network to some extent. However, the decrease of anonymity depends on the connectivity of the exit router as well as on the connectivity of its adjacent routers. Even for the low bandwidth routers, connectivity at a given point in time can be as high as 120-300. For routers from the set of 10% fastest routers, the connectivity may be higher than 1500. Thus, exploiting Tor topology at just one point in time may not be sufficient. A much more efficient way would be to observe Tor connectivity changes over time. Indeed, an application that requires a persistent connection, will force the routers in the circuit to maintain a connection between them for the application's lifetime at least. An attacker who wants to trace such a communication needs to observe the exit node for a while and eliminate routers which it looses connections to. On the other hand, if user's application drops a connection, an attacker may observe a new defect in the topology and link this defect with the user's application (note that if the attacker controls the exit node, she can cause the connection to drop.) In this way, we come to a simple but powerful idea: observation of local Tor network connectivity dynamics gives us a way to decrease the anonymity provided by Tor. More specifically, to trace long-lived (or persistent) connections and to reveal short-lived connections.

3.1 Canonical Connectivity Scanning

We will now show how an attacker can scan a Tor relay to find out what TLS connections are established between it and other relays. To explain how this works, we first have to delve into details of the Tor specification. In order to prevent an attacker to force a relay to open a new TLS connection for each extend request, a Tor relay uses an existing connection (if any) corresponding to the fingerprint specified in the extend request no matter what IP address was indicated. This could potentially allow a malicious party to perform a man-in-the-middle attack. For the two relays R_1, R_2, the attacker would send an extend request with a forged IP address X to R_1 before other circuits (and hence a connection) are established between R_1 and R_2. If the machine at IP address

X were then to connect to R_2 and forward all of the traffic it received from R_1 to R_2 and vice versa, it could perform a byte-counting attack. To prevent this from happening, Tor uses a countermeasure called *"canonical connections"*. Briefly, a connection to a router is canonical if the destination IP address of this connection corresponds to the one in the consensus. If a Tor relay gets an extend request with a fingerprint, it should use an existing canonical connection corresponding to this fingerprint.

We noticed that Canonical connections give an attacker a convenient way to determine how routers in the Tor network are connected to each other. When sending a `RELAY EXTEND` cell, the circuit originator specifies both the identity fingerprint and the IP address of the router he wants to extend the circuit to. Assume that the attacker wants to figure out whether a router A is connected to a router B. In order to do this, the attacker forges a Tor `RELAY EXTEND` cell with the fingerprint of router B and `127.0.0.1` with an unreachable port (port 1 for example) and sends it to router A. When the cell is received, the reaction of router A depends on whether it has a connection to router B:

– If A has a canonical connection to B (it should be noted that if a connection exists it is almost always canonical), router A ignores the IP address from the forged `RELAY_EXTEND` cell and uses the already established TLS connection, extends the circuit and sends back `RELAY_EXTENDED` cell.
– If A does not have a connection to B then it tries to make a new TLS connection using the address from the received cell. Obviously, the connection attempt is refused which causes router A to send a `DESTROY` cell to the attacker.

By inspecting the cell the attacker receives back from router A, she can determine whether router A is connected to router B. Evidently, the attacker can probe router A for connection with any router contained in the consensus[3].

3.2 Connectivity probing via timing attacks

We now consider a second, somewhat less powerful approach for determining whether two relays are already connected. When a client extends a circuit from relay R_1 to relay R_2, the time until he received the `RELAY EXTENDED` reply from R_2 depends on whether a TLS connection between R_1 and R_2 is already set up or whether it needs to be established first. In the later case, both the additional network and the cryptographic latency are considerable.

A TLS connection setup between Tor relays can cause huge delays, especially if version 2 or above of the handshake protocol is used. This delay is caused by network latency and the large number of protocol steps until the `CREATE` cell can be sent (see Figure 2 for details). If a TLS connection needs to be set up to create a circuit, a delay on the order of 7.5 round-trip times is added to the

[3] By coincidence, this scanning technique can not only be used to scan the connectivity of a Tor router, but also to scan for open ports on random IP addresses from a relay that has an all-reject exit-policy.

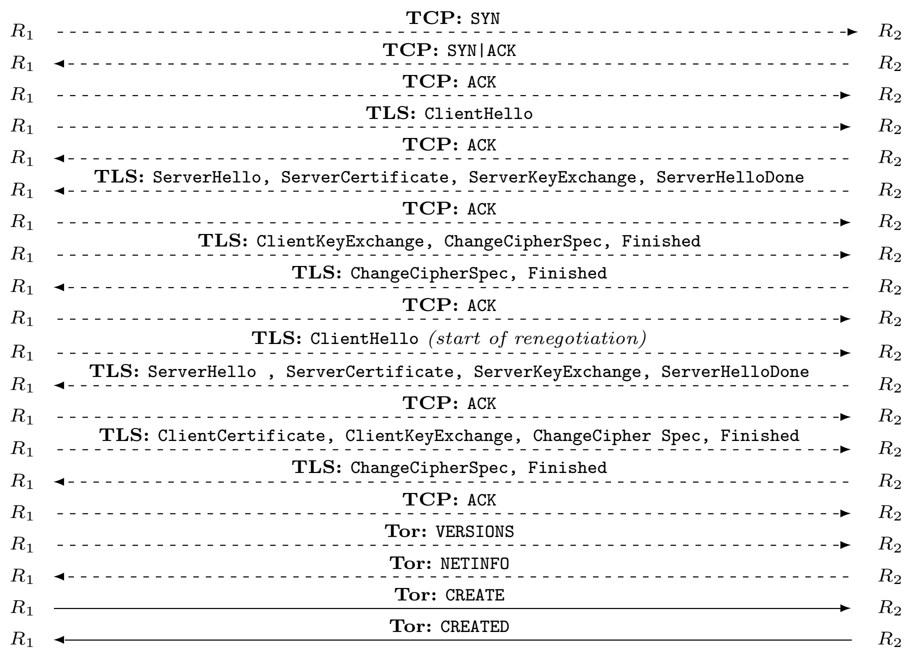
R1
R2
TCP: SYN
TCP: SYN|ACK
TCP: ACK
TLS: ClientHello
TCP: ACK
TLS: ServerHello, ServerCertificate, ServerKeyExchange, ServerHelloDone
TCP: ACK
TLS: ClientKeyExchange, ChangeCipherSpec, Finished
TLS: ChangeCipherSpec, Finished
TCP: ACK
TLS: ClientHello (start of renegotiation)
TLS: ServerHello , ServerCertificate, ServerKeyExchange, ServerHelloDone
TCP: ACK
TLS: ClientCertificate, ClientKeyExchange, ChangeCipher Spec, Finished
TLS: ChangeCipherSpec, Finished
TCP: ACK
Tor: VERSIONS
Tor: NETINFO
Tor: CREATE
Tor: CREATED

Fig. 2. Tor circuit setup. The last two steps are performed always. Steps marked with dashed lines are performed only when there is no TLS-connection between R_1 and R_2.

circuit creation until the `CREATE` cell is received by R_1. Approximately 6.5 round trips are required for the TLS connection setup alone, another round-trip for the v2 handshake. By sending multiple `RELAY EXTEND` requests and comparing the time it takes for the first one to arrive versus subsequent ones, we can determine whether a relay is connected to another relay. This has been confirmed with experiments. The disadvantage of this method is that network jitter as well as cell forwarding delays by the relay scanned can add significant amounts of noise which makes the method less reliable. Moreover, in contrast to the method described in the previous subsection, this method will really establish TLS connections to all routers that are scanned and not just prolong the lifetimes of the connections that are already open.

4 Attacking Tor using connectivity dynamics

4.1 Tracing long-lived streams

Tor is used by many people to establish long-lived SSH sessions, download very large files (sometimes using file-sharing applications, even though this is frowned upon) and to communicate over instant messaging networks. The latter usage of Tor is particularly important for countries with repressive regimes such as

China, Iran, or Syria: people are regularly sent to prison or worse for statements critical of their government. The use-cases described above imply long-lived TCP-streams which necessarily create long-lived TLS-connections between Tor routers which are used to carry the stream. Thus, we show how an attacker knowing the exit node of a long-lived TCP-stream can link it with the guard node using our scanning techniques[4].

One-Hop Attack In this attack, we assume that the attacker controls one or more very fast exit routers which see a significant fraction of the traffic exiting the Tor network, thus she gets access to pseudonyms of the users (ex. cookies, logins). This is not an unrealistic scenario; some organizations have control over sizable portions of the total exit traffic: according to the consensus current at the time of writing this paper, 7.2% of total exit capacity were provided by the Chaos Computer Club, 5.9% by Torservers.net and 5.4% by Formless Networking LLC. The attacker is curious to connect the pseudonyms with the guard triplets for the users that pass through her Exit relays. Assume that one of the attackers' nodes E (see Figure 3) is selected as the exit node of a circuit. By looking at the traffic pattern, the attacker will be able to infer that the connection to the exit node is likely to be of long-lived type. The attacker then starts the attack:

1. The attacker starts scanning the middle node M for connectivity using either of the techniques described in the previous section. The set of connected nodes necessarily includes the guard node G in question and makes up its initial anonymity set.
2. Next, the attacker continues with the connectivity scanning of the middle node for several hour or even days in hope that the majority of the nodes of the initial anonymity set will disconnect (nodes with dash lines on Figure 3.)
3. The attack stops when the anonymity set of the guard node is considerably reduced or when the user closes the long-lived TCP-stream.

When the attack is finished, the user's guard node will be contained in the resulting anonymity set (node G and another node with the solid line on Figure 3) along with some number of other connections that can be considered as "noise". The attacker may also infer extra information from the speed of the connection, which will indicate whether the middle or the guard node are the bottleneck for the traffic of the long-lived circuit; this helps her to further shrink the set of candidates for the guard node since it allows to discard very active routers from the list of candidate guard nodes.

[4] One important note is that in the current Tor protocol, the connections between two routers which last more than 7 days are marked as "bad" for new circuits and no new circuits can be added to such connections. However persistent circuits inside these connections are not closed and will continue running. At the same time we cannot see these persistent OR connections anymore using our probing techniques after 7 days have elapsed.

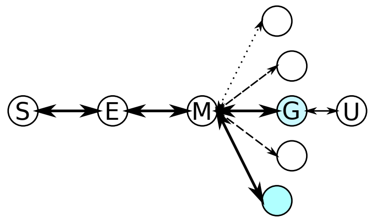

Fig. 3. One-hop attack against long-lived connections

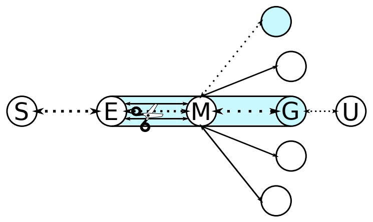

Fig. 4. Differential scanning attack

Two-Hop Attack This attack does not required from the attacker to control any relays in the Tor network and can be performed by a server (or an attacker close to the server) who tries to reveal the guard nodes of pseudonymous users connecting to the server. The attack starts from connectivity scanning of the exit node (similar to one-hop attack) in order to reduce the anonymity set of the middle node. After having narrowed down the set sufficiently, the candidate middle nodes are scanned resulting in the anonymity set of the guard node. The attack might be successful if either middle or guard nodes are low-bandwidth which might be inferred from the connection latency by the attacker. We also assume that exit node is medium or low-bandwidth. The difficulty in the two-hop attack comes from the fact that many middle nodes reachable from the exit node would come from a set of active routers with many connections. This will result in hundreds of candidate guard nodes even after several days of scanning. This effect happens due to "immortal"connections formed between active routers, which we will describe in Section 5. In spite of its simplicity, the described attack is quite powerful since:

(i) it does not require control over any relays in the Tor network. The attacker merely probes relays (probing could be also done from a distributed set of addresses);
(ii) it is cheap in terms of bandwidth: in order to scan one router the aggregated amount of traffic that needs to be sent and received is less than 5 MBytes (for the current size of 3000 routers in Tor network);
(iii) it is fast: the average time of scanning one router is 20 seconds and scanning of different routers can be easily parallelized.

Experimental results In order to estimate how efficient the attacks can be in the wild, we used Python to implement a rudimentary Tor client which provides basic functionality. The client can establish a TLS connection to an arbitrary Tor router, complete Diffie-Hellman key establishment protocol and send and receive Tor relay cells. In other words, the client is able to create and extend arbitrary chosen circuits. Using canonical connectivity scanning, our client is able to check a Tor router for connectivity with 99% of other routers in the Tor network in less than 30 seconds.

In order to check the correctness of the proposed canonical connectivity scanning, we scanned two routers under our control `omicron` and `Layercake` for five days from February 11th until February 16th, 2012. During the experiment the routers had bandwidth weights in the range [500 - 1500] for `omicron` and in the range [15000-55000] for `layercake` which means that the later was in the top 10% set of fastest and thus most frequently chosen routers. Both relays had Guard flags and did not have Exit flags. Since the routers were operated by us, we could gather the real time statistics directly from them using the Tor control port. We then compared the results from the canonical connectivity scan and from the control port. Figure 5 shows the number of persistently connected Tor routers over time, i.e. those routers which were connected to our routers at the start of the experiment and never disconnected during the experiment. The close match of the results as shown on Figure 5 demonstrates that canonical connections scanning provides reliable results. The slight difference in the results is explained by the difference of scanning frequency: for canonical connection scanning, each sample cannot be taken faster than every three minutes (i.e. the lifetime of an idle Tor TLS connection); the data from the routers control port however was fetched every ten seconds. According to Figure 5, for the router with bandwidth weight 1500 (`omicron`), the number of persistently connected routers decayed from 303 to 20 in just 12 hours. This matches with our prediction from Section 5.1. It then took 4 days for another 18 routers to disconnect. Among the remaining two connections, there was one which we established by ourselves and which we tried to identify. The decay rate of persistent connections of the high-bandwidth router (layercake) looks similar: the number of persistent connections drops sharply from 1116 to 300 in 12 hours and then decays slowly. We tested canonical connection scanning against several Tor routers not under our control. The result for one such router with bandwidth weight in range [2040-2190] is shown on Figure 6. We observed a very similar behaviour: a big chunk of connections drop quickly, and then it decays slowly. After two days of scanning, we found 12 persistent connections.

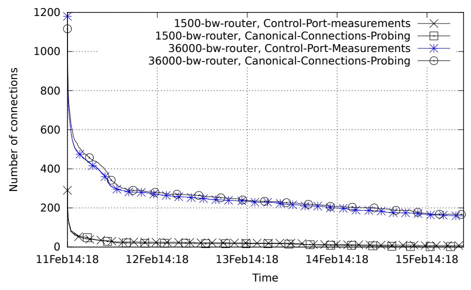

Fig. 5. Decay rate of Persistent connections. Canonical connection scan vs. direct measurements

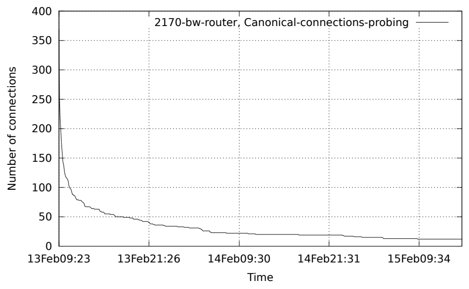

Fig. 6. Persistent connections decay rate for a random router

4.2 Differential scan attack

Attack description Consider user which periodically checks some Web server or a web service that instructs the user's browser to periodically re-establish streams. Google Mail for instance builds a series of short-lived (around 2 minutes) TCP sessions. Another example are news web sites with auto-refresh contents. In this section, we describe an attack on such kind of recurrent connections. The aim of the attacker is to find at least one of the guard nodes of a pseudonymous user (identified by a cookie or a login credential) that uses such a service for several days. Note that this attack does not require a single long-lived circuit or session. It just requires that a Tor client is connected to the Tor network for non-negligible amount of time within the span of a month (as long as the guards are still valid).

Similar to Section 4.1, in this attack, the attacker has control over a significant fraction of the exit capacity of the Tor network. Assume that a user visits a Web server S (see Figure 4) that causes recurrent connections to occur. Ten minutes after the first connection, his initial circuit should expire and the user's Tor client will try to build a new circuit. Given a sufficient number of exit nodes controlled by the attacker, the circuit will include one of the attacker's exit nodes E. Once the exit node receives incoming traffic destined to the web server it executes the following sequence of steps:

1. The exit node E observing the stream to the web server determines the middle node M of the circuit that caused the stream to be established and transmits it to the attacker.
2. The attacker probes the connectivity of M and remembers the list of routers connected to it (nodes connected to M both with dash and solid lines on Figure 4).
3. E sends a `DESTROY` cell[5] down the circuit which leads to the circuit termination. The circuit termination may lead to the connection termination between the middle node and the user's guard node with some probability which can be estimated using expressions from Section 5.2.
4. The attacker waits for three minutes and starts the scan of M again.
5. The attacker computes the difference between the sets obtained via the first and the second scans, i.e. he determines connections which were present in the first list but absent in the second (node G and another node with dash line.) We say that we have a differential with node G and M if G is in the difference.
6. The attacker then repeats steps 1-3 each time one of her exit nodes is chosen for the recurrent connection.
7. Once an attacker has performed the above steps often enough, and given that the circuit closure event caused the connections closure frequently, she can derive the user's three guard nodes: the probability of having the guard node in the difference should converge to 1/3.

[5] if the attacker wants to be more stealthy she can just wait until the circuit expires by itself

This attack may be further enhanced by scanning the full network at regular and frequent intervals. Then if the connection to the malicious Exit arrives shortly after the full network scan, the attacker will have additional differential connectivity information in order to filter the noise. Our experiments have shown that the full network scan can be done in 3 minutes using 20 hosts (using Amazon EC2 service, a day of full network scans with 3 minutes between scans costs around 80 USD).

A similar but less stealthy approach can be used to track any users connection. Assume that a user connecting to a server chose one of the attackers exit node. This allows the attacker to incorporate a small piece of code in each HTML document requested by the user, which artificially creates recurrent connections. Specifically the user can be redirected to an arbitrary address and port. Note that in the current Tor network, aggregated exit bandwidth for different port is different, thus by choosing the appropriate port range, the attacker can increase the probability that her exit node is chosen: at the time of the experiment total exit capacity was approximately $5 \cdot 10^6$ Kbytes/s, the bandwidth capacity of scarce ports[6] was about $1.2 \cdot 10^6$ Kbytes/s.

Experimental results We have implemented a proof of concept version of our differential scanning technique and have tested it using sets of paths generated by a modified version of the Tor client – this client does not create any circuits but simply outputs randomly generated paths with user-specified constraints. These paths are then used to build circuits through the control port of the Tor daemon. After a circuit has been built, a scan is conducted, then the circuit is torn down, the program waits for 200 seconds and scans again. To perform experiments more quickly we have implemented this in a parallelized manner on Amazon's EC2 platform so that many (non-interfering) experiments can be conducted in parallel. As a first experiment, we used only one guard node with capacity of 36500 and allowed for middle nodes with capacity of 1600 or lower in the consensus[7]. For 150 paths, 125 successful differential scans were performed. In these, the guard node we had selected appeared at position 1 in the list of most frequently occurring guard relays in the difference sets, having been counted 58 times.

1. `C37B234FAD013453B90375EB55864FEBC876104A`: 58 `(PPrivCom052)` bw=36500
2. `CA1CF70F4E6AF9172E6E743AC5F1E918FFE2B476`: 35 `(spfTOR3)` bw=29800
3. `0B7ED44C67DBE50313F0B32BD335D093D0474CE8`: 33 `(bauruine2)` bw=117000
4. `847B1F850344D7876491A54892F904934E4EB85D`: 31 `(tor26)` bw=20
5. `DB8C6D8E0D51A42BDDA81A9B8A735B41B2CF95D1`: 30 `(rainbowwarrior)` bw=81300
6. `173B220F9F32F39086D5661274A47485EDA26131`: 29 `(TorExitProgressbar9)` bw=650
7. `1603DFE9FC373ECDA39046FADB5A76B87A4BA36B`: 27 `(StickItToTheMan)` bw=46800
8. `1F52D692FA2C21B23FAD4D711A7BF17BAE2673DF`: 26 `(alice)` bw=7170
9. `47916CAB5878C810E7EF71A316D37FC823CC7F52`: 26 `(CCN)` bw=53100
10. `95A0D58710EA9B61DAD3A01CAD3BE77DACA76BEF`: 25 `(OccupyMyPants)` bw=30300

This shows that differential probing works in practice: there's a drastic reduction in the anonymity set of the guard nodes, even for high capacity guard

[6] There are several scarce ports still usable by Web browsers

[7] See Section 5 for justification of the choice of the bandwidths. In brief: (1) the product of bandwidths of the guard node and middle node should not exceed 300 million to avoid "immortal connections ; (2) the attack works best when either the guard or the middle node are not high-bandwidth.

nodes. Below is the concrete data of one of the experiments in which we had chosen guards of capacity 300, 412, and 501, constrained the capacity of the middle nodes to 30,000 and scanned different middle nodes in 134 trials[8]:

1. `A58E0F05C1939725D7247BA60BA3135DB88209BC`: 43 (`jef0lewkia`), bw = 501
2. `D3378ABA009078158DB59E8B36B8EBB88B309BA7`: 40 (`torn0t`), bw = 412
3. `2629979FD21BF3B522E818B73F6F8D0B5D8A5CF0`: 40 (`tapir`), bw = 300
4. `A9C039A5FD02FCA06303DCFAABE25C5912C63B26`: 29 (`chaoscomputerclub5`), bw = 173000
5. `FA486415B86D28CD047D10F76768E4E88A182F71`: 28 (`ZhangPoland1`), bw = 56400
6. `131B60B9AFE6AEA60042132D648798534ABEA07E`: 28 (`wagtail`), bw = 24400
7. `4536ED68D9DB4B2FF532AD43A632AAF600B798CC`: 27 (`Unnamed`), bw = 116
8. `1D8625690AB9729FB2040D8194EC0D6789A4D092`: 25 (`TOR1CINIPAC`), bw = 43900
9. `FC35DE87F6E4022693323275F6B8EE5F72FD21B5`: 24 (`Unzane`), bw = 3160
10. `CA1CF70F4E6AF9172E6E743AC5F1E918FFE2B476`: 23 (`spfTOR3`), bw = 28700

Again, although we have some spurious low-bandwidth routers in the top ten, these results show that the attack described above works well in practice. Note that in real life, the attacker will perform scans for any circuit which has been detected to be established by a unique pseudonym of a user and for which the middle node is below a certain threshold bandwidth.

We now try to estimate how many measurements the attacker should make when low capacity guards are being used. There are 1,440 minutes in a day; this means that if the attacker is unlucky (i.e. his exit is not selected and then she needs to wait for 10 minutes until the circuit expires in order to get another chance) there are 144 measurement chances per day. The fact that attacker controlling a fraction f of the exit bandwidth tears down circuits to which she gets access, increases the number of measurement slots available to the attacker by a factor $\frac{1}{1-f}$, which for $f = 1/3$ results in $144\frac{f}{1-f} = 72$ slots. If the upper bound for the capacity of the middle node is set to 30000 then (according to Figure 12) there is about 40% chance for a circuit to go through such middle node. This reduces the amount of measurements to 29 per day. The attacker will continue the attack until he obtains about 40 measurements, which means the attack will run for about 1 day. Note that the attack is very successful if the bandwidth of one of the user guard nodes is below 500. There is about 3% chance that a user's client has chosen a guard node with low capacity, i.e. $G_{min} < 500$, into his triplet of guard nodes. Thus this attack could affect more than 10,000 daily users of the Tor network.

5 Analysis of the attacks

5.1 Long-lived connections

In section 4.1, one could notice that after a relatively short period of scanning time, a substantial number of routers which were connected at the beginning of the scan disconnected. The decay rate of the number of persistently connected routers becomes very low after. In other words, when the number of connections drops to some value, the reduction rate of the anonymity set of the guard node an attacker is trying to identify becomes negligible. This value can be considered

[8] `jef0lewkia` was involved in 43 circuits, `torn0t` in 45 and `tapir` in 46.

as a threshold for this attack which we try to estimate in this section. There are two main reasons of why a connection between a pair of routers may last very long and thus increase the anonymity set:

1. this pair of routers is a part of a long-lived circuit similar to the one the attacker tries to identify, i.e. a circuit which is used for an application which requires a long-lived TCP-stream;
2. the circuit creation rate over this connection is high and there is always at least one circuit inside this connection which prevents it from closing. Such *immortal* connections form if the product of bandwidths of the two routers exceeds a certain threshold as will be shown below.

First, we estimate the number of connection of the first type. Figures 7 and 8 show circuit duration distributions over a connection between two high bandwidth routers (`layercake` with bandwidth weight of 35300 and `bouazizi` with bandwidth weight of 69700 for 13 of Feb 2012). Figures 9 and 10 show circuit duration distributions over a connection between a high bandwidth router and a non-high bandwidth router (`omicron` with bandwidth 491 for 13 of Feb 2012 and `layercake`). Life-times of circuits have two clear peaks at around 10 and 60 minutes due to properties of the Tor protocol: renewal time of "dirty" circuits and the lifetime of "clean" circuits which have never been marked as "dirty". According to the measurement, circuits with life-time longer than 2 hours constitute less than 1.5% of the total number of circuits. From this we can assume that the majority of long-lived connections in Tor are of the seconds type, i.e. formed by high circuits creation rate over these connections. Another observation is that the anonymity set of persistent streams is small compared to the anonymity set of non-persistent streams.

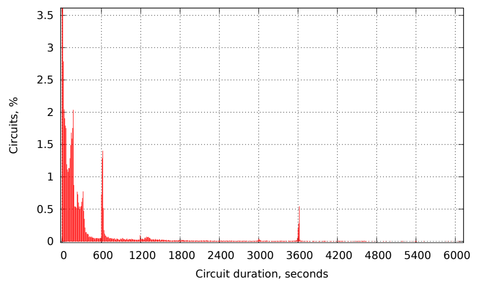

Fig. 7. Circuit duration probability density function between two high bandwidth routers

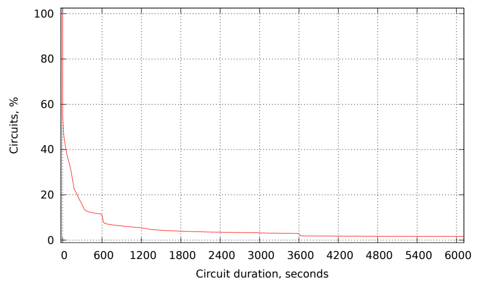

Fig. 8. Circuit duration distribution function between two high bandwidth routers

To estimate the number of long-lived connections of the second type, we observed several active (i.e. high-bandwidth) guard Tor routers under our control and measured client circuits arrival rate. Figure 11 shows the number of new

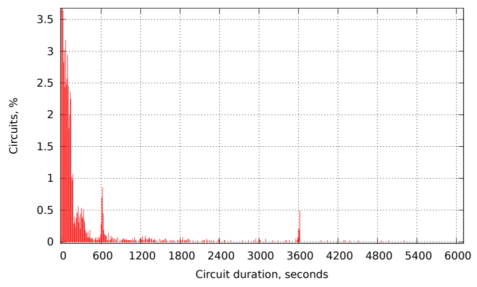

Fig. 9. Circuit duration probability density function between a high bandwidth and non-high-bandwidth routers

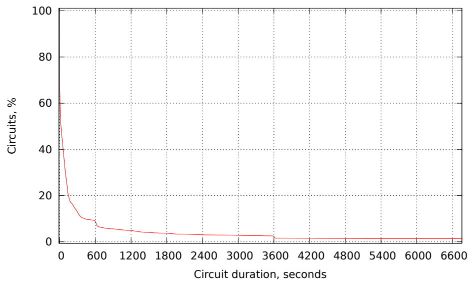

Fig. 10. Circuit duration distribution function between a high bandwidth and non-high-bandwidth routers

circuit per ten seconds gathered during two days on one of our active routers. We observed that:

- Circuits arrive according to the non-homogeneous Poisson process.
- Assuming that client circuit arrival rate is proportional to the guard router's bandwidth, we estimate an average circuit arrival rate R in the whole Tor network to be about 900 circuits per second (not at peak times). In the expressions below one can also use the value of circuit arrival rate for the specific time of the day instead of the average value.
- The average circuit duration time t_{avg} is about 200 seconds which varies only slightly for routers with different bandwidth weights.

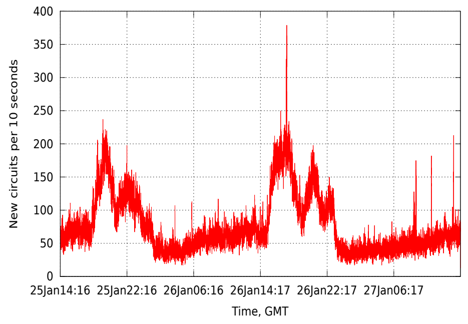

Fig. 11. Circuit arrival rate for an active high bandwidth router

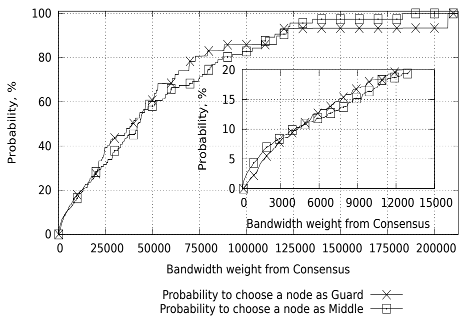

Fig. 12. Probability for a node to be chosen as a guard and a middle node

We now estimate the probability that a pair of routers A and B is connected with almost immortal connection. Note that a TLS-connection between Tor relays is closed only if no circuits were carried over this connection for three minutes. In other words, for a connection to stay open, the time between arrivals of two consecutive circuits should not exceed the average circuit duration plus 3 minutes. Denote by Δt the time of the attack. Then during this time, $\Delta t \cdot R \cdot p_{a,b}$ new circuits will arrive. Here $p_{a,b}$ is the probability of routers A and B to form an edge in a new circuit[9].

$$p_{a,b} = 2 \cdot \frac{bw_a bw_b}{bw_{total}} \left(\frac{1}{bw_{guards}} + \frac{1}{bw_{exit}} \right),$$

where bw_{guards} is the total bandwidth of guard nodes, bw_{exit} is the total bandwidth of exit nodes, bw_{total} is the total bandwidth of the whole Tor network, bw_a and bw_b are bandwidths of routers A and B respectively. Taking into account that circuits arrive according to the Poisson distribution, the probability to have an "immortal connection" can be computed using the following expression:

$$P_{immortal}(A,B) = (1 - e^{-R \cdot (t_{avg} + t_{idle}) \cdot p_{a,b}})^{\Delta t \cdot R \cdot p_{a,b}},$$

where t_{idle}= 180 seconds. A connection between A and B almost never closes if $P_{immortal}(A,B)$ is close to 1. Using this expression we find that immortal connections are formed between routers of bandwidth $> 17,500$ (or routers with product of bandwidths above 300 million). By bandwidth we mean not the advertised bandwidth but actual figures from the Consensus computed by Tor authorities' bandwidth measurements and used in the Tor code to choose routers for the circuits. Given the bandwidth of a router, an attacker can estimate the number of immortal connections that it has and decide whether it is worthwhile to perform the attack.

Figure 14 shows complementary cumulative bandwidth distribution of Tor relays along with the share (i.e. the percentage of total number of Tor relays) of persistent connections for each bandwidth[10]. Note that bandwidth distribution of Tor relays changed only slightly during March and February 2012. For example, if an attacker decides to scan a Tor relay with bandwidth weight of 5000, she can expect that this relay has about 1% of "immortal" connections. Given 3000 Tor relays, this yields the anonymity set of 30 relays. This kind of prediction corresponds well with the experimental results obtained in section 4.1. If $bw <$ 1300, the attack should give the unique solution[11]. Note that although only few routers have large percentage of immortal connections, these routers are high-bandwidth and and are selected more frequently.

[9] This expression for $p_{a,b}$ is an approximation since it does not take into account all peculiarities of the Tor path selection algorithm, in particular, the expression ignores weights which are assigned to a relay based on its position in the circuit and its flags. We compared our approximation with the precise calculation and found that simpler approximation is sufficient for our purposes and makes the analysis easier to understand.

[10] Note that bandwidth distribution can be approximated by the Pareto distribution with minimal value $x_m = 350$ and exponent $\alpha = 0.85$.

[11] For 11th of February 17:00, 2012, there were 2388 nodes out of 2897 with bandwidth less than 1300. Their aggregated capacity was 371,159 out of 9,458,556 total capacity of the whole Tor network.

In order to give a first order approximation of how long we should wait until a persistent connection is detectable among other "non-immortal" connections, we collected connection duration statistics from Tor routers operated by us for 7 days[12]. Figure 13 shows the connection duration distribution for two pairs of routers: medium-to-medium bandwidth (lower curve, in green), medium-to-high bandwidth (in red). For medium-to-high only 5% of connections have duration of more than three hours. In the case of medium-to-medium bandwidth routers (see Fig. 13), only 5% of connections between them have duration of more than 1 hour. In ten hours, 99% of all non-immortal connections should disconnect for both cases. Thus, we expect that if a persistent connection under observation has a duration of more then 10 hours, the probability of its successful identification depends mostly on the number of immortal connections.

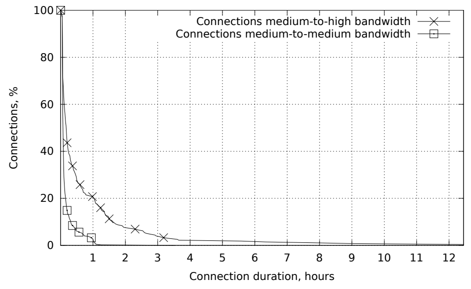

Fig. 13. Connection duration distribution

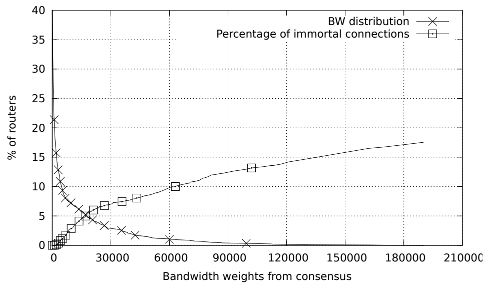

Fig. 14. Tor bandwidth distribution and share of immortal connections

5.2 Differential scanning attack

In this section, we explore the limits of the differential scan attack. Assume that an attacker tries to reveal a guard node g by observing circuits $\{c_1, ..., c_k\}$ which leads to scanning of a set of middle nodes $M = \{m_{c_1}, m_{c_2}, ..., m_{c_k}\}$. Let T denote the set of all Tor relays and $|T| = n$. Then we define $d : M \times T \longrightarrow \{0, 1\}$ in the following way:

$$d(m_{c_i}, r) = \begin{cases} 1 & \text{if we observed a differential between Tor relays } m_{c_i} \text{ and } r \text{ for circuit } c_i \\ 0 & \text{otherwise.} \end{cases}$$

The success of the attack depends on: (1) $Signal = \sum_{i=1}^{k} d(m_{c_i}, g)$, i.e. number of differentials with guard node g , and (2) $Noise_{r_j} = \sum_{i=1}^{k} d(m_{c_i}, r_j)$, number of differentials with some other Tor relay r_j, $j = 1, ..., n$. We then use signal-to-noise ratio $SNR = \frac{Signal}{\max_j\{Noise_{r_j}\}}$ as a measure of the success of the attack.

[12] The logs we obtained were stored on computers with full-disk encryption behind the firewall of our academic institution.

We first estimate the Signal and Prob$[d(m_{c_i}, g) = 1]$. Denote by t_0 the time when c_i was destroyed. $d(m_{c_i}, g_1) = 1$ iff the connection which carried c_i closes 3 minutes after c_i is destroyed. This happens if no new circuit with duration t arrives during $[t_0 - t; t_0]$ and no circuits arrive during $[t_0; t_0 + t_{idle}]$. Let $f(t)$ be the probability density distribution of the circuit duration. Then given that the circuits arrive according Poisson distribution, we have:

$$\text{Prob}[d(m_{c_i}, g_1) = 1] = e^{-\int_0^\infty R \cdot p_{a,b} \cdot t \cdot f(t) \mathrm{d}t} \cdot e^{-R \cdot p_{a,b} \cdot t_{idle}} = e^{-R \cdot p_{a,b} \cdot (t_{avg} + t_{idle})}, \tag{1}$$

where R is the current circuit arrival rate of the whole Tor network, and $p_{a,b}$ is the probability of router A and B to form an edge in a circuit (see Section 5.1).

To estimate the *Noise* and Prob$[d(m_{c_i}, r) = 1]$ for some Tor relay $r \neq g$ we use the following approach: $d(m_{c_i}, r) = 1$ if: (a) at the time of the first scan, there is a connection between m_{c_i} and r; (b) there is no connection at the time of the second scan. We find the probability of the second event using (1). We compute the probability of the first event as the ratio of the average gap between connections and the average duration of the connection. Note that the average delay between two circuits given that it is larger than $t_{avg} + t_{idle}$ is computed as:

$$I = \frac{\int_{t_{avg}+t_{idle}}^{\infty} t \cdot \lambda_{a,b} \cdot e^{-\lambda_{a,b} \cdot t} \mathrm{d}t}{e^{\lambda_{a,b} \cdot (t_{avg}+t_{idle})}} = t_{avg} + t_{idle} + \frac{1}{\lambda_{a,b}}, \tag{2}$$

where $\lambda_{a,b} = R \cdot p_{a,b}$. Thus, the probability that there is a connection between A and B at an arbitrary point of time is:

$$1 - e^{-\lambda_{a,b} \cdot (t_{avg}+t_{idle})} \cdot \frac{I - (t_{avg} + t_{idle})}{I} = 1 - \frac{e^{-\lambda_{a,b} \cdot (t_{avg}+t_{idle})}}{\lambda_{a,b}(t_{avg} + t_{idle}) + 1}$$

Thus:

$$\text{Prob}[d(m, r) = 1] = \left(1 - \frac{e^{-\lambda_{a,b} \cdot (t_{avg}+t_{idle})}}{\lambda_{a,b}(t_{avg} + t_{idle}) + 1}\right) \cdot e^{-\lambda_{a,b} \cdot (t_{avg}+t_{idle})}$$

To demonstrate how the above expressions work, we used the set of 125 middle nodes from the experiment described in Section 4.2 with bandwidth weights equal or less then 1600. Figure 15 shows: (a) the Signal of the guard node against its bandwidth. (b) the Noise of a Tor relay against its bandwidth. As can be seen from the figure, for low-bandwidth nodes the signal is close to its maximum value. This happens since for this type of node, the probability that the connection between it and a middle node carries just one circuit is very high. Low circuit arrival rate of a low-bandwidth relay also implies the low value of noise since the probability to have a connection between it and a middle node is low.

6 Discussion and potential countermeasures

In this paper, we have shown two ways to extract topology information of the Tor network. One way to determine the real connectivity of Tor relays is to

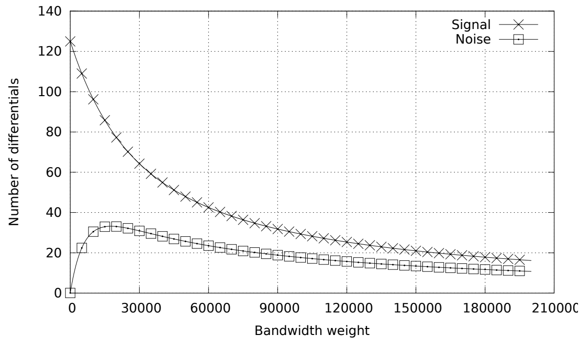

Fig. 15. Signal and Noise for differential scan

exploit a Tor countermeasure against man-in-the-middle attacks called canonical connections. This method is cheap but can be eliminated in future versions of Tor by changing the specification. A potential countermeasure against the canonical connection scan is to abolish canonical connections. Of course this must be done while preserving the circuit multiplexing feature. An obvious approach is to not have a set of connections that are identified by the fingerprint as a primary key but rather by both the fingerprint and the IP address of the relay. This prevents our attack, but needs to be weighed against the fact that incorporating this fix gives an attacker a new way to perform denial-of-service by resource exhaustion against Tor relays.

A different approach for measuring relay connectivity is to use timing information of the connection establishment as a side channel: circuit extension by one hop takes much less time if the link on this hop already exists. This method is less robust then the one exploiting canonical connections, but at the same time the countermeasures are not straightforward; experiences in side-channel cryptanalysis have shown that simple countermeasures like adding randomized delays can often be defeated. At the same time, a fully connected graph for the Tor network – i.e. having each relay connected to all the other relays at all times – probably is too expensive from a performance standpoint. The balance to strike here is to add sufficient noise to make timing attacks unreliable to attackers.

Information on the network topology obtained by either means can be the basis of two efficient and practical attacks which were presented in Sections 4.1 and 4.2. The first attack can be used to reduce the set of possible guard nodes of a user who uses long-lived connections (e.g. SSH sessions). The second attack is based on recurrent connections and can be used by a malicious server to determine the tuple of guard nodes selected by the client of a user.

Additionally, since our connectivity revealing techniques are orthogonal to the existing attacks described in the literature, they can be used to improve many of them substantially. Indeed, during the times when the number of Tor routers was small, several attacks were available to adversaries. These attacks allowed to link the exit and entry nodes of a user's circuit. However, once the number of Tor routers grew, those attacks became too expensive in terms of required bandwidth and time. This is because for those attacks to be successful, exhaustive probing of each link in the Tor network was required. Given a way to determine the real connectivity of Tor network, these attacks can become practical again since the amount of links to be probed is significantly reduced.

7 Conclusion

All prior research on Tor assumed opacity of the Tor network topology – meaning that the attacker had to assume a fully connected graph. In practice, the real degree of a node in this graph is substantially smaller than its maximum at any given point in time. For the first time, we have shown methods to determine the real connectivity of relays in the Tor network and the dynamics of the topology of the whole Tor network. Based on this, we described several novel attacks that use this information to deanonymize the entry points of the users into the Tor network.

8 Acknowledgement

We would like to thank anonymous reviewers for numerous useful comments.

References

1. Back, A., Möller, U., and Stiglic, A. Traffic analysis attacks and trade-offs in anonymity providing systems. In *Proceedings of the 4th International Workshop on Information Hiding* (London, UK, UK, 2001), IHW '01, Springer-Verlag, pp. 245–257.
2. Bissias, G. D., Liberatore, M., Jensen, D., and Levine, B. N. Privacy vulnerabilities in encrypted http streams. In *In Proceedings of Privacy Enhancing Technologies Workshop (PET 2005* (2005), pp. 1–11.
3. Chaum, D. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM 24*, 2 (1981), 84–88.
4. Danezis, G. The traffic analysis of continuous-time mixes. In *In Proceedings of Privacy Enhancing Technologies workshop (PET 2004), LNCS* (2004), pp. 35–50.
5. Goldschlag, D. M., Reed, M. G., and Syverson, P. F. Hiding routing information. In *Information Hiding 1996* (1996), R. J. Anderson, Ed., vol. 1174 of *Lecture Notes in Computer Science*, Springer, pp. 137–150.
6. Levine, B. N., Reiter, M. K., Wang, C., and Wright, M. Timing attacks in low-latency mix systems. In *Proceedings of Financial Crypto 2004* (2004), vol. 3110 of *LNCS*, Springer, pp. 251–265.
7. Manils, P., Chaabane, A., le Blond, S., Kaafar, M., Castelluccia, C., Legout, A., and Dabbous, W. Compromising tor anonymity exploiting p2p information leakage. In *Technical Report 00471556, INRIA, April 2010. http://arxiv.org/abs/1004.1461.*
8. Murdoch, S. J., and Danezis, G. Low-cost traffic analysis of Tor. In *In Proceedings of the 2005 IEEE Symposium on Security and Privacy. IEEE CS* (2005), pp. 183–195.

9. Panchenko, A., Niessen, L., , and Zinnen, A. Website fingerprinting in onion routing based anonymization networks. ACM, pp. 1–10.
10. Serjantov, A., and Sewell, P. Passive attack analysis for connection-based anonymity systems. In *In Proceedings of European Symposium on Research in Computer Security (ESORICS* (2003), pp. 116–131.
11. Wang, X., Chen, S., and Jajodia, S. Network flow watermarking attack on low-latency anonymous communication systems. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2007), SP '07, IEEE Computer Society, pp. 116–130.
12. Wang, X., and Reeves, D. S. Robust correlation of encrypted attack traffic through stepping stones by manipulation of interpacket delays. In *Proceedings of the 10th ACM conference on Computer and communications security* (New York, NY, USA, 2003), CCS '03, ACM, pp. 20–29.
13. Yu, W., Fu, X., Graham, S., Xuan, D., and Zhao, W. Dsss-based flow marking technique for invisible traceback. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2007), SP '07, IEEE Computer Society, pp. 18–32.
14. Zhu, Y., Fu, X., Graham, B., Bettati, R., and Zhao, W. On flow correlation attacks and countermeasures in mix networks. In *in Proceedings of Privacy Enhancing Technologies workshop* (2004), pp. 26–28.