Finding the guard: leveraging bandwidth statistics to identify TOR guard nodes

Maarten Dorrestijn
s4665198

Supervisor: Jaap-Henk Hoepman
Second reader: Greg Alpar

20-06-2019

Abstract

In this thesis, we present an attack which aims to identify the guard node of a hidden service. The TOR network aims to protect the anonymity of hidden services on the network. The guard nodes play a vital role in protecting this anonymity, and identifying the guard node is a vital step in breaking the anonymity of a hidden service. The attack is based on leveraging the bandwidth statistics of the guard nodes in the TOR network to identify attack traffic that is sent to the hidden service. By inducing patterns in the published bandwidth statistics, the aim is to identify which guard node is used by the hidden service. Different methods of identifying the patterns are tested and evaluated, along with an evaluation of the requirements of the amount of attack data that is required to induce a measurable pattern.

Table of Contents

1. Introduction

In the information age, anonymity and privacy seem to become more and more important in modern societies. As the recent introduction of the General Data Protection Regulation has shown, countries are becoming more and more aware that the privacy of their citizens is something that should not be taken lightly. It is therefore also not strange that many technical projects have been undertaken which in some way aim to increase the privacy of its users. The Onion Router, or the TOR project is one of these projects. The network is continually being developed and the scientific community has researched the network for quite some time. As such, uncovering potential flaws in the design of TOR has also been the subject of much academic research, and this in turn has lead to a large number of vulnerabilities being found in TOR over time.[1] In this thesis we will continue on this path, and demonstrate a guard identification attack on hidden service.

In this thesis we will demonstrate an attack which leverages a vulnerability in the TOR network. We will first give a general description of TOR and a description of the presented attack(§2), and then discus some work that has already been done with regards to the proposed type of attack(§3). Next, we will give an in-depth description of some of the inner workings of the TOR network which are relevant to the attack(§4), which will allow for a formal definition of the problem at hand to be given (§5). After this formal definition has been stated, we will discus some insights in the requirements of the attack (§6). Following this, we will show and discus the final test setup(§7) and the results of the tests (§8). After the results have been discussed, we will outline possible countermeasures against the demonstrated weakness(§9). At the ending we will discus some future lines of work(§10) and give a conclusion with regards to the proposed attack(§11).

1 Some examples of these attacks can be found in [1],[2],[3],[5].

2. General overview of TOR

2.1 The TOR network

The TOR network is a network of nodes which aims to provide anonymity to its users. It achieves this by routing traffic through a number of nodes before sending the traffic to the final destination. By doing so, the TOR network allows its users to mask their true identity while using the Internet. The TOR network is run by volunteers who all contribute a node to the network. These nodes all work together in routing the data of the TOR users. When data is sent through the TOR network, a virtual 'circuit' is created through which the data is routed. These circuits are the pathways of data through the network, and commonly consists of three nodes. In these circuits, each node is sent the identity of the next node in the circuit. If data is sent through the circuit, each node passes that data along to the next node, and the last node sends the data to the final destination, commonly an HTTP server or any other server the users wishes to communicate with. The "layered" encryption makes sure that the data is encrypted several times using shared keys with each node in the circuit. Each node that receives the data, 'peels' off a layer of encryption and passes the data along. Using this mechanism, the data needs to be routed through each node in the network to be readable for the final destination. By using this scheme, the final destination will have no idea what the identity of the actual sender of the data was. And regarding the nodes, each node in the circuit will only know the identity of the node or user from which the data was received and the identity of the node to which the data needs to be sent. No single node is able to correlate the identity of the user with the destination of the data the users sends.

An extension can be made on this default procedure to not only allow for a user hide its identity from the endpoint, but to also allow an endpoint to hide its identity from its users. Such a hidden endpoint is called a "Hidden Service" in the TOR jargon. A Hidden Service works by having the endpoint choose a number of introduction nodes in the network. The Hidden Service constructs a circuit to these introduction nodes and published the identity of these introduction nodes in a set of publicly known nodes known as the "Hidden Service Directories." If a user wishes to communicate with a Hidden Service, it requests the list of introduction nodes from a Hidden Service Directory and constructs a circuit to one of these nodes. A cryptographic handshake is performed through these introduction nodes, and the user sends the identity of yet another node to the Hidden Service, which will be the rendezvous node[9]. Both the user and the Hidden Service set up a circuit to the rendezvous node, and can begin to communicate with each other.

An important consideration regarding the anonymity of the TOR network is the trustworthiness of the first node in the network a user or a hidden service connects to, commonly called the "guard node", since this is the only node in the network which knows the true identity of the end user or hidden service. No matter how compromised the rest of the network might be, if the guard node remains uncompromised, this user will remain anonymous. Since a users anonymity relies on the trustworthiness of the guard node, the TOR network defaults to the use of a single guard node for all

circuits of a particular user. This guard node is then pinned for a period of approximately six months. If many different guard nodes are used for the circuit creation, then the chances that one of them is malicious and identifies the user becomes much larger than when only a single guard node is used. Either this node is compromised and the user loses their anonymity, or the node is playing by the rules and the users anonymity is protected for each and every circuit. However, if an attacker were to learn which guard node is used by a user, the identity of the user might still be retrieved by compromising the guard node. Because the TOR network also aims to defend its users against state-level actors, it is not unthinkable that a government agency might seize a guard node in order to identify a particular user. Due to this, the TOR network rotates the guard nodes approximately once every six months in order to make identifying the guard node more difficult[4].[2] These two effects a balance in the amount of guard nodes that need to be trusted. On one side the number of guard nodes that are used needs to be as low as possible to avoid malicious ones, and on the other side is it also important that a guard node isn't used for too long in order to avoid detection.

One aspect of the TOR network is that besides providing anonymity for its users, it also aims to promote scientific research in the network.[3] Due to this, the TOR network has a built-in system through which nodes publish some data regarding their status.[4] The data that is published through this system includes, amongst other things, the amount of traffic that a node has handled on behalf of the TOR network per day.

2.2 The proposed attack

The attack that will be discussed in this thesis will focus on identifying the guard node of a given hidden service. To do so, the bandwidth statistics published by each node of the TOR network will be leveraged to identify said guard node. This attack will work by sending large volumes of data to a hidden service, and monitoring the bandwidth statistics of all the guard nodes in the TOR network. The guard node that is used by the targeted hidden service will handle all this extra traffic, and increase its bandwidth statistics accordingly. By detecting spikes in the bandwidth statistics the correct guard node could be found.

2 To see the TOR developers take on this problem, see https://trac.torproject.org/projects/tor/ticket/8240.
3 The TOR project has commented multiple times on the importance of scientific research into the network, see for example https://blog.torproject.org/how-do-effective-and-impactful-tor-research.
4 These statistics can be accessed through https://metrics.torproject.org.

3. Related work

3.1 Guard node detection

The TOR network has been the subject of a number of academic studies which revealed various weaknesses in the network.[5] These weaknesses have usually been centered around the anonymity of the users. Over the course of several years, there have been multiple attacks exploiting these weaknesses. Within these attacks, a distinction can be made between attacks which target TOR clients, TOR hidden services or both. To the attacks that focus on hidden services, the guard nodes are the most interesting aspect of the network. Due to the fact that the guard node of a hidden service is the only node within the TOR network that knows the actual identity of a hidden service, identifying the guard node of a hidden service is usually a first step to uncovering the identity of a hidden service. From the perspective of a law enforcement agency, uncovering the guard node might be enough to uncover the identity of the hidden service, since the government-attributed power to view or subpoena the traffic of the guard node can be enough to learn the identity of the hidden service.

As such, there are a number of attacks which focus on uncovering the guard node of a hidden service. The most commonly known one is the sybil attack[7]. While the sybil attack is actually a name for a whole set of attacks using sybil nodes, one of these attacks allows for the discovery of the guard node of a hidden service. Sybil nodes are nodes which are controlled by a single operator and aim to get as much information and influence in the network as possible. The guard discovery attack using sybil nodes works by adding a set of middle nodes to the TOR network and creating a huge amount of circuits to the target hidden service. As will be explained in §4.4, the chance that one of the circuits from the hidden service to the rendezvous-node is build through a sybil node increases with every built circuit. By applying one of the many end-to-end correlation techniques[5], it is possible to determine if such a circuit is build through a sybil node and if so, to determine what the guard node of the hidden service is. This attack can currently be mitigated by the optional vanguard-addon[14].

3.2 Guard node detection through Denial-Of-Service

3.2.1 Sniper attack

The attack that will be discussed in this thesis will stem from a different kind of attack than the sybil attack. Since the cornerstone of the attack will be that measurable effects occur in guard nodes when a certain amount of data is passed through it, the attack is more related to guard node attacks through denial-of-service.

In 2014, the *sniper attack* was published[3]. The sniper attack abused a buffering mechanism of TOR which allowed an attacker to have a TOR node use an arbitrary amount of memory. The researchers

5 Some examples of these attacks can be found in [1],[2],[3],[5].

showed that this could lead to the Operating System of the TOR node to kill the TOR process, which effectively allowed an attacker to shut down a TOR node. The sniper attack had therefore multiple applications. It could simply be used to take down certain TOR nodes, or it could be applied to uncover the guard node of a hidden service. The discovery of guard nodes through the sniper attack is possible by taking out random nodes of the network while inspecting the data flow from the attacker to the target hidden service. If the sniper attack takes down the guard node of the hidden service, the hidden service will be unreachable and will stop responding to the attacker. Using this method, a guard node discovery attack could be performed on the TOR network.

3.2.2 Congestion attack

One method of retrieving the guard node of hidden service is through congesting the guard node. A variant of such a clogging attack was published by Evans et al[2]. This variant sends a lot of data through randomly selected TOR nodes and observers whether or not the link to the hidden service is affected. Deploying such an attack however, requires a lot of resource from the attacker. The attacker needs to be able to send enough data to have a measurable impact on each node in order for the effects to become visible on the link between the attacker and the hidden service. Also, another important aspect of the attack is that the target of the attack can easily detect whether or not the attack has occurred, simply by monitoring the reachability of the guard node.

3.2.3 Bandwidth statistics analysis attack

Rochet et al showed in 2018 that there was another way of discovering the guard node of a target hidden service[1]. Their research showed that the bandwidth statistics published by each node could be exploited to identify the guard node of a hidden service. Their attack worked by sending the maximum possible amount of traffic to the hidden service which would increase the bandwidth used by the guard node. This would mean that the guard node published unusual high bandwidth statistics, which would identify the guard node. The TOR network responded to the research by updating the default time period over which the statistics would be published from 4 hours to 24 hours. By doing this, the total bandwidth requirements of the attacker increased sixfold, and the practical application of this attack is severely hindered. The method employed by Rochet et al was based on the congestion of the guard node by sending as much data as possible. Due to the increase in measurement period, this attack requires a very large capacity from the attacker. The attack that will be demonstrated in this thesis will differ from this attack in that the demonstrated attack will attempt to minimize the amount of data used by the attacker.

4. In-depth analysis of TOR

4.1 TOR Components

Given that the TOR network contains a large number of complex components, it is necessary to describe the workings of the components that are relevant for the attack.

4.2 Directory authorities

The TOR network is ‘governed’ by 9 directory authorities which are tasked by generating a series of parameters through which the TOR clients can use the network. The directory authorities publish these parameters through a document called the network status consensus[10]. These directory authorities are a key piece of infrastructure for the TOR network and it uses this network status consensus document to distribute the relevant information regarding the network to the clients. The network status consensus contains a list of all registered TOR nodes in the network, together with some information regarding these nodes. One element of the network status consensus is the bandwidth consensus, which will be discussed in §4.3. The information that is published regarding these nodes contain a set of flags. These flags indicate multiple things regarding the nodes,and can mark, amongst other things, a node as a guard node or as an exit node. The ‘guard’ and ‘exit’ flags mark that a node can be respectively chosen as a guard node or as an exit node. What these flags mean exactly is explained in §4.3.2.

4.3 Bandwidth consensus

4.3.1 The function of the bandwidth consensus

The TOR network uses a system called the bandwidth consensus to regulate the amount of traffic each node receives. Because not all nodes have equal capacity, it is important to make sure that nodes that can handle more traffic than other nodes also receive more traffic. The bandwidth consensus gives each node a score, the bandwidth consensus score. This score is used in the path selection algorithm, as described in §4.3.2 to determine the chance that a node is used in a path. The bandwidth consensus score of each node is determined by the bandwidth authorities. The bandwidth authorities are nodes of the TOR network which continuously perform bandwidth throughput tests on the nodes of the network and measure how much data a node can handle. These bandwidth authorities then publish their results to the directory authorities which use that data to calculate the final consensus score[12].

4.3.2 The path selection algorithm

By using the published bandwidth consensus the client is able to pick nodes for the circuits it creates. Based on the bandwidth consensus scores a weight is attributed to a node during a random pick of a node. However, the bandwidth consensus score is not directly used as the weight. The final weight that is given to a node when randomly picking one is dependent on the flags a node has and the position a

node is picked for in a circuit. The TOR network differentiates between three positions in a circuit: guard, middle and exit. The path selection algorithm has certain modifiers which affect the weight of a node with certain flags in certain position. These modifiers are used to determine, for example, the chance that a guard node is chosen for a middle position. Using these modifiers, the TOR network can regulate what kind of traffic is received by guard, middle and exit nodes. The modifiers are published in the network status consensus. The path selection algorithm first determines the role for which it will select a node. For each node it will then multiply the bandwidth consensus score with the applicable modifier to determine the weight. The modifiers are published for nodes that have a guard flag, an exit flag, both guard and exit flags or neither of these flags. For each of these category, the modifiers are published for each possible role, with the roles being the guard, middle or exit node. By doing so, the TOR network can regulate which type of nodes are selected for which role. If there is a shortage of exit nodes, the network will make sure that no exit nodes are used for the middle position. The directory authorities will then set the modifier for nodes flagged as ‘exit’ for the middle position to 0. At the moment of writing, the TOR network has indeed a shortage of exit nodes, and as such, nodes that have the ‘exit’ flag are not allowed to be guard nodes in order to save bandwidth for exit traffic[13].

4.3.3 The influence of the bandwidth consensus

As follows from the path selection algorithm as described in §4.3.2, the bandwidth consensus score of a node determines the amount of traffic a node will receive. As such, it is possible to make a rough estimation of the bandwidth usage of a node based on the consensus score. By being able to determine the influence of the bandwidth consensus, it is possible to filter some noise from the bandwidth statistics by subtracting the traffic that is generated by regular clients based on the consensus score. Because the attack that we will present in this thesis will be based on detecting the influence of attack data on the bandwidth statistics, the attack could be more precise if certain factors could be accounted for. As such, determining the influence of the bandwidth consensus on the amount of traffic a node receives is an important step in helping to identify the influence of attack traffic by removing noise from the statistics.

However, before it is possible to determine the exact influence of the bandwidth consensus score on the amount of traffic a node receives, an important distinction must first be made. The data usage of TOR nodes can be divided into two categories: the data used for circuits in which the node is a guard node, and the data used for circuits in which it isn’t a guard node. The distinction between these two categories is important because the data produced by the circuits in which the node is a guard node is almost independent from fluctuations in the consensus score, due to the longevity of a guard node/client relation. The data produced by circuits in which it isn’t a guard node is dependent on fluctuations in the consensus score due to the fact that node selection is done for every new circuit, and node selection is done randomly with weights based on the consensus score.

In order to determine this ratio between consensus score and non-guard traffic it is only possible to analyze nodes which are flagged as neither guard nor exit. The reason why it isn’t possible to analyze a

node flagged as guard is because its bandwidth statistics will also contain guard traffic. The reason why it isn't possible to analyze a node flagged as exit is somewhat more complex. As stated, the consensus at the moment of writing disallows a node flagged as both exit node and guard node to function as guard node. Due to this, the final attack will only focus on nodes which have only the guard-flag. Because of the different amount of middle nodes and exit nodes and the fact that an exit node could play the role of a middle node but not vice-versa, there are more nodes which can play the middle-role than nodes that can play the exit-role. As such, it is only possible to infer the ratio of non-guard traffic that flows through nodes with only the guard flag compared to their consensus weight by analyzing that ratio in nodes having neither the guard nor the exit flag. At the time of writing, the modifier for guard-flagged nodes playing the middle role is 4061, while the modifier for middle nodes playing the middle role is 10000,[6] which would imply that the ratio of non-guard traffic flowing through a guard node compared to its consensus weight is 2.5 times as low as it is for a middle node.

To calculate the ratio between the non-guard traffic flowing through middle nodes compared to its consensus weight, the total amount of traffic flowing through the middle nodes will be divided by the total amount of consensus score these nodes have. To calculate the effect of the consensus score, we will use historical data published by the TOR through the CollecTor project. The data used will be from September 2018. The average bandwidth usage of middle nodes during that period was: 2022 MB/s . The average total amount of consensus score of the middle nodes during this period was: 10110000.[7] This leads to the conclusion that on average, middle nodes receive 0.000200 MB/s per consensus point. By applying the modifiers for the guard and middle position we have that guard nodes will receive:

$$\frac{4061}{10000} \times 0.000200 = 8.12\text{e-}5$$

8.12e-5 MB/s per consensus point.

4.4 Effect of attack data on the network

The way a circuit is constructed to a Hidden Service is also relevant to the proposed attack of this thesis. The attack builds on the fact that if traffic is sent to a hidden service, the guard node traffic will pass through the guard node. However, the traffic will of course also pass through other nodes, and it is important to make sure that the attack data is spread out across as many nodes as possible. By doing so, there will be as little as possible influence on the bandwidth statistics of non-target guard nodes. An important aspect of contacting a hidden service is the rendezvous node. Through the Hidden Service Authorities and the introduction nodes the hidden service and the client can construct a circuit. In this process, the client is able to pick an arbitrary node in the network as the rendezvous node. The

6 These values are obtained through the current network status document provided by CollecTor, see [13].
7 Both these figures are calculated on the archived data of CollecTor.

rendezvous node is the node to which both the hidden service and the client will construct a circuit. By spreading the attack data over many circuits which use different rendezvous nodes, it is possible to also spread the data across many different nodes in the TOR network. Only the guard node will receive all the traffic in this manner. The tool published by the researchers of the original paper implements this method[1].

4.5 Effect of attack data on nodes

Because the proposed attack will have an active nature, it must be determined in which way the network is influenced. As was shown in §4.4, the bandwidth will be spread fairly equally amongst all nodes in the network and the target guard node will receive all attack data. However, the underlying assumption of the attack is that the attack data will directly add to the amount of traffic used by the guard node, and not, for example, ‘push’ other traffic away. This assumption is based on multiple factors. First, as shown in §4.3, the chance that a node is used in a circuit is based on its consensus weight. As such, the amount of traffic that TOR clients will request through a node is based on the ratio between consensus score and MB/s as calculated in §4.3.3. This implies that even though the consensus score might be influenced by the attack data, this effect will be proportional to the ratio between consensus score and MB/s as calculated and can thus be compensated for. However, as Rochet et al argue, there is also a large chance that extra attack data will not influence the capacities of the guard node at all. They argue that due to the fact that over 80% of the guard nodes run at only 50% of their available capacity, guard nodes have a substantial capacity buffer. Considering the fact that the proposed attack will attempt to minimize the amount of attack traffic, it is very likely that the amount of attack data will not exceed the available bandwidth and will thus not have any measurable impact on a guard node.

To further demonstrate this point, a test has been performed on the real TOR network. A node was selected as the guard node for our own hidden service and two attack pulses of 0,5 MB/s send through the guard node with both pulses lasting two days. The node had an average bandwidth usage of 4 MB/s, which implied that we would use 12,5% of it in order to make sure that the node was not saturated. The aim of the test was to see whether or not the attack data would also increase the bandwidth statistics by 0,5 MB/s. If the final produced bandwidth statistics would show an increase which was significantly lower than 0,5 MB/s, then it would imply that our extra attack data had an unforeseen effect on the node. If the increase was close to the amount of attack data, then no additional effects would be present. In the figure below, the node ‘ParkBenchInd001’ with fingerprint ‘F10BDE279AE71515DDCCCC61DC19AC8765F8A3CC’ was used. Measurement zero is the measurement that started on 2019-02-13 20:00 UTC.

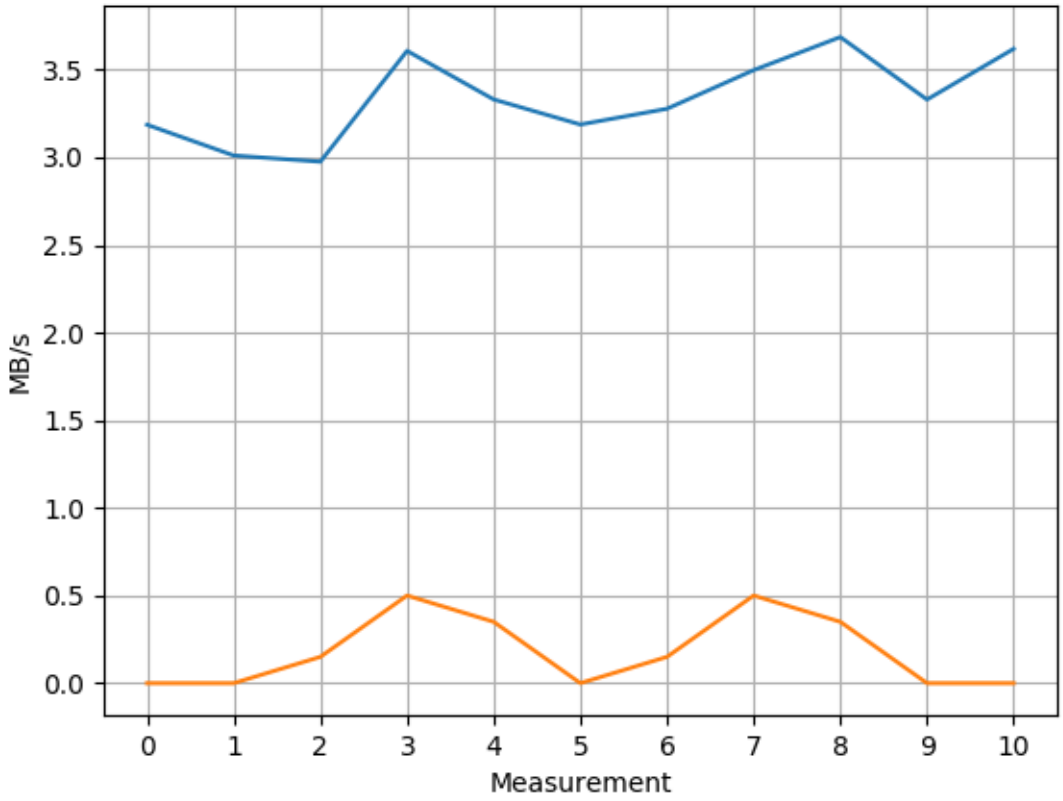

Figure 1

In the first figure, the blue line represents the bandwidth statistics corrected for the consensus weight. All the statistics have been reduced according to the average consensus weight, and 8.12e-5 MB/s per consensus point have been subtracted. The bandwidth statistics are corrected for the consensus weight to remove the non-guard traffic and remove a source of noise. The yellow line represents the amount of attack traffic that was sent through the guard node during the measurement. As can be seen, a reasonable correlation can be shown between the amount of attack traffic and the bandwidth statistics. During measurement 3, a nice increase of 0,6 MB/s can be seen relative to measurement 1, the last measurement during which no attack data was sent. Also, the bandwidth statistics seem to follow the decrease in attack data from measurement 4 also nicely. However, measurement 7 only shows an 0.25 MB/s increase when compared to measurement 5.

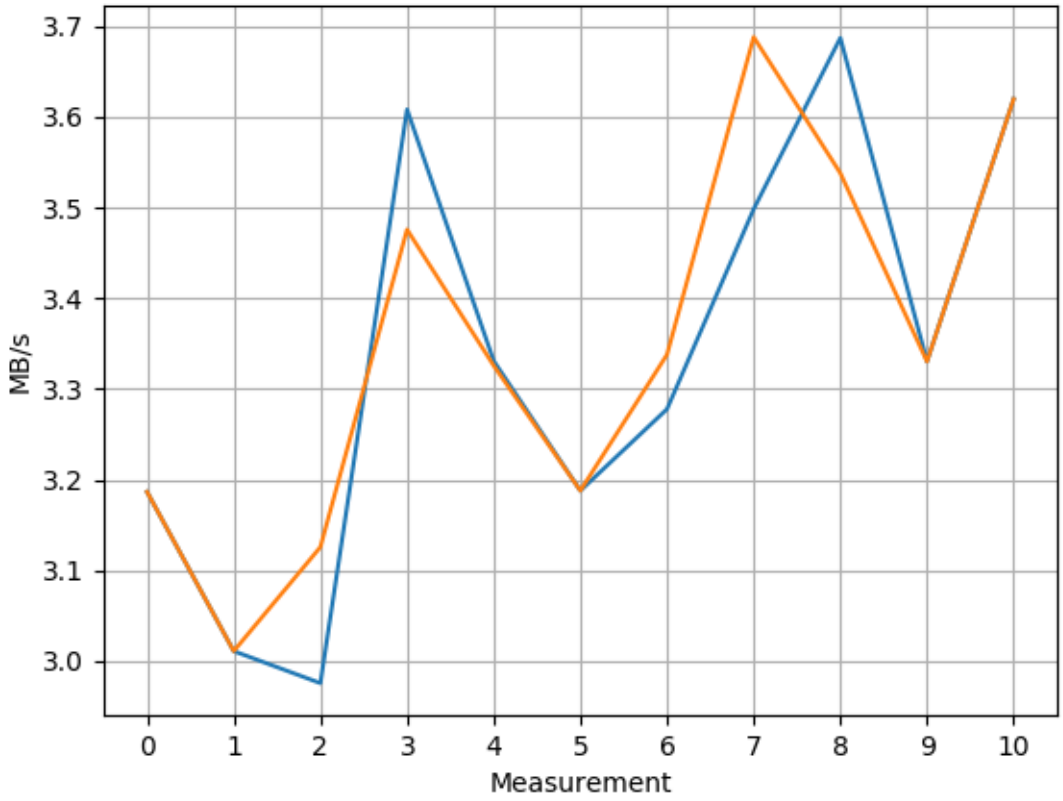

Figure 2

In the second figure, the null-measurements 1 and 5 are taken as reference, because those are the latest measurements during which no attack data was sent. The yellow line shows the expected value of the bandwidth statistics by taking the amount of attack data sent during each measurement and adding it to the last null-measurement. The blue line still represents the actual measurements. As can be seen, the yellow line seems to match rather well with the blue line. The bandwidth statistics at measurement 3 and 8 are both 0.15 MB/s too high, and the measurements at 2 and 7 are both 0.18 MB/s too low. Given the fluctuations in bandwidth usage which is native to the TOR network, these differences are not unexpected. The fact that the bandwidth measurements are about equally too high as they are too low implies that these differences are produced by the randomness of the bandwidth statistic data, and not some obvious unwanted effect.

We can further show the absence of any unwanted side effect by comparing the amount of non-attack traffic during measurements which were influenced by attack data with measurements that are not. The non-attack traffic of the node during statistics which were influenced by attack data can be obtained by taking the compensated statistics and subtracting the amount of attack data that was sent through the node during that statistic. Table 2 shows the values of various measurements which were done two days before and after the attack period (2019-02-13 until 2019-02-24). Table 1 shows the measurements

which were affected by attack data and shows the amount of non-attack traffic for these measurements. When we compare the mean values of both these groups, we find that they only differ 0.0041 MB/s. Due to the fact that the means differ as little as they do, it is reasonable to state that no external effect can be observed which influenced the bandwidth statistics during this test.

Start date of statistic	Compensated non-attack traffic
2019-02-15 20:00	2.8256 MB/s
2019-02-16 20:00	3.1080 MB/s
2019-02-17 20:00	2.9809 MB/s
2019-02-19 20:00	3.1277 MB/s
2019-02-20 20:00	2.9982 MB/s
2019-02-21 20:00	3.3368 MB/s

Table 1: The non-attack traffic during attack-influenced statistics (mean value: 3.0966 MB/s)

Start date of statistic	Compensated value
2019-02-11 20:00	3.0744 MB/s
2019-02-12 20:00	2.5046 MB/s
2019-02-13 20:00	3.1865 MB/s
2019-02-14 20:00	3.0110 MB/s
2019-02-18 20:00	3.1878 MB/s
2019-02-22 20:00	3.3302 MB/s
2019-02-23 20:00	3.6195 MB/s
2019-02-24 20:00	3.2285 MB/s
2019-02-25 20:00	2.6904 MB/s

Table 2: The uninfluenced statistics (mean value: 3.0925 MB/s)

While this test is nowhere near a thorough analysis of the exact effects attack data might have on an arbitrary guard node, it does show that no obvious effects seem to exist. As such, the main argument for the assumption that extra attack data will have no other measurable effect other than an increase in the bandwidth statistics does not depend on the results of this test, but it is reinforced by it. Given that the path selection algorithm determines which nodes get used in circuits and the path selection algorithm uses nothing else than the consensus score, and the arguments made by Rochet et al that it is very unlikely that guard nodes will be taxed above their capacity, and the results of this tests, it is very likely that the assumption that there will be no measurable unwanted effects of the extra attack data on the target guard nodes holds.

4.6 Guard rotation period

A final consideration with regards to the technicalities of the TOR network is the amount of time that a hidden service uses the same guard node. Currently, this is a random number chosen between 30 days and 365 days, meaning that on average each hidden service will chance its guard nodes once every half year. This implies that any practical attack will need to have a timespan which is significantly lower than this half year, because if a change of the guard happens during the attack time frame, all the gathered data before the chance is incompatible with the data after the chance. Also note that when the guard changes, it is chosen according to the path selection algorithm as described in §4.3.

5. Formal definition

5.1 Defining the model

In order to properly state the problem, an formal definition will be given of the behavior of the relevant components of the TOR network for the attack. Using this definition, it will be possible to properly assess the various requirements of the attack. We will first define some functions which model various aspects of the network, then show how these functions are related, and then express the problem statement in these functions. For practical reasons, the time frames that are used are rounded to hours.

5.2 Definitions

5.2.1 Definitions concerning TOR

The following definitions described the working of the TOR network:

-Let G be the set of guard nodes.
-Let *IsTargetGuard(g)* be the proposition that a guard node is the guard node of the target hidden service.
-Let *Published(g,t*) be the published bandwidth statistics for guard node *g* in time interval *t.*
-Let *Traffic(p)* be the value of measured traffic of a bandwidth statistic.
-Let *Bandwidth(g,p)* be the bandwidth used by guard node *g* in the time interval over which statistic *p* was measured by all non-attacker traffic, note that no direct data is available about this traffic.
-Let *GuardBandwidth(g,p)* be the bandwidth used by a guard node for guard traffic in the time interval over which statistic *p* was measured.
-Let *NonGuardBandwidth(g,p)* be the bandwidth used by a guard node for non-guard traffic in the time interval over which statistic *p* was measured.
-Let *Consensus(g,p)* be the published consensus average in the time interval over which statistic *p* was measured.

5.2.3 Definitions concerning the attacker

Given these functions concerning the TOR network, the attacker will determine the following functions:

-Let *Attack(t)* be the function that determines the amount of traffic the attacker sends at each moment in the time interval *t.*
-Let *Detect(g,t)* be the function that decides if node *g* is the guard node that the attacker is looking for in the given time frame *t*. This function will always be based on *Published(g,t).*
-Let *MaxAttack(t)* be the maximum amount of traffic send per second by *Attack(t)* during time interval *t*.

-Let *AttackInfluence(p)* be the amount of bandwidth send by the attacker in the time interval over which a statistic *p* was measured.

5.3 Prerequisites

Using these definitions, various components of the TOR network can be linked together. First of all, it follows that given a time frame *t:*

1:

$$\forall g \in G(\forall p \in published(g,t)(Bandwidth(g,p) = GuardBandwidth(g,p) + NonGuardBandwidth(g,p)))$$

Also, as discussed in §4.3.3, it follows that the non-guard traffic through a node is linear proportional to the published consensus score, and thus we have again for a given time frame *t*:

2: $\forall g \in G(\forall p \in published(g,t)(NonGuardBandwidth(g,p) = Ci * Consensus(g,p)))$

For *Ci* the ratio between published consensus weight and actual traffic as calculated in §4.3.3.

Another aspect of the definitions is that for a given time attack frame *t*:

3:

$$\forall g \in G(IsTargetGuard(g) \Leftrightarrow (\forall p \in published(g,t)(Traffic(p) = Bandwidth(g,p) + AttackInfluence(p))))$$

5.4 Problem statement

Using the above definitions and relations between the defined functions, we have that the main problem which we will research will be:

Find Detect(g) and Attack(t) such that for all g and the chosen attack time interval t:

$$P(Detect(g,t) \mid IsTargetGuard(g))$$

is maximized, while

$$P(Detect(g,t) \mid \neg IsTargetGuard(g))$$

is minimized and

$$Attack(t)$$

is minimized.[8]

8 What “minimized” means for *Attack(t)* is discussed further in §6.2.1.

6. Requirements of the *Attack* and *Detect* functions

6.1 Considerations regarding the *Detect(g,t)* function

6.1.1 Purpose of the *Detect(g,t)* function

The *Detect(g,t)* function, as shown in the formal definition, will evaluate the likelihood that a guard node is the target guard node based on the bandwidth statistics produced by the guard node. As such, the *Detect* function will function as a classifier function, which determines for each guard whether or not the guard node will be marked as the target guard node. It is important to note that such an approach implies that the *Detect* function could possibly mark multiple nodes in the guard set as the target guard node.

6.1.2 Handling the randomness of the input

The main aspect of the *Detect*(*g,t)* function is that it will discriminate between nodes which have received the attack data and nodes that have not. The main difference between nodes that have received attack data and those that have not is that the former will have higher bandwidth usages. However, since the statistics produced by the nodes are up to some degree random, there will be many nodes that will have higher than usual bandwidth usage during the time frame of the attack. The *Detect(g,t)* function will have to account for this fact. One way to deal with this problem is to measure the relevance of a perceived increase in bandwidth usage within the light of the randomness of other data of the node. For a node with larger swings in bandwidth usage it could be more difficult to attribute an increase in bandwidth usage to the *Attack* function and by accounting for this fact, the *Detect* function could be more precise.

6.1.3 The statistical nature of the input

One method of dealing with the level of randomness of the input is by considering the nature of the data. As has been shown before, the traffic that travels through a TOR node consists of the data of many different circuits. One important aspect of having the total bandwidth usage being the sum of many smaller processes is that the *central limit theorem* can be applied. The central limit theorem states that the sum of independent random variables tends towards a normal distribution[6]. By having the total bandwidth usage being the sum of all individual circuits that are build through a node, the bandwidth statistics should tend towards a normal distribution. However, due to the specific characteristics of the traffic usage of a node as explained in section §4.3, assuming a normal distribution should be done carefully. One statistical tool that might help with classifying the data is *student's t-test*[6]. This statistical procedure works on data which follows *student's t-distribution.* The *t-distribution* is the distribution in which a limited number of samples are available from a dataset which would converge to a normal distribution. This weaker variant of the normal distribution allows the use

of the *t-test*, a test which determines the likelihood that two samples are produced in the same way. This *t-test* accounts for many of the problems concerning the randomness of the input.

6.2 Considerations regarding the *Attack(t)* function

6.2.1 Minimization of *Attack(t)*

As shown in the formal definition, the problem focuses amongst other things on minimizing the *Attack(t)* function. The reason that this aspect of the problem is relevant is because part of the current defenses against the attack published by Rochet et al is based on detecting sudden large volumes of traffic and alerting the end user.[9] The assumption in which this defense is based is that it will only be possible to identify a guard node based on sudden spikes induced by the attacker in the bandwidth statistics of that guard node. However, using a more sophisticated *Detect(g,t)* function, it might be possible to work with very little traffic.

However, given the nature of the *Attack(t)* function, it is still not entirely clear when the *Attack*(t) function is minimized. One intuitive metric would be measuring the total amount of data sent by the *Attack(t)* function. However, as stated above, it is also relevant to keep the amount of data that is sent per second to a minimum. Multiple variants with different peak values and different amounts of total throughput will be researched.

6.2.2 Requirements of *Attack(t)*

One important consideration of the *Attack(t)* function is that the function should send the attack data in such a way that it creates data points in the bandwidth statistics of the target guard node which can be meaningfully compared to each other, and that the comparison should be able to distinguish between sets of data points influenced by the *Attack(t)* function and sets that are not. As such, it is likely that an *Attack(t)* function which sends a constant amount of traffic through the guard node for the duration of the interval will not generate meaningful results. Since comparing the data points in the set created by the target guard node after applying such an *Attack(t)* function will show that all data points are generated in the same way, they will not be distinguishable from the set of data points generated by the non-target guard node with a larger average amount of traffic usage. Only when the increase is so high that it could be detected by comparing the published traffic usage to an expected value based on other factors, most notably the bandwidth consensus. However, such a method would require much more attack traffic than when using previous bandwidth statistics, since only a part of the traffic can be reliably predicted by the bandwidth consensus as described in §4.3.3 and such a method will thus be less accurate.

It is important to note that this does not imply that the *Attack(t)* function cannot be a single pulse, merely that the final set of published statistics in *t* should include data points which are affected to different degrees of attack data.

9 The vanguard addon provides this functionality

As such, it is important to see that the *Attack(t)* function will need to alternate to some extent. This way, it will be possible to get differences between the data points and establish whether or not the data points were influenced by the *Attack(t)* function.

6.2.3 Effects of an alternating *Attack(t)*

By taking an alternating *Attack(t)*, some side effects will affect the final published bandwidth statistics. One important side effect is what will be referred to as the ‘tainting’ of certain published measurements. Due to the fact that the measurements done by nodes are not done synchronously, the beginning of a high pulse generated by *Attack(t)* will for many nodes fall in the middle of a measurement. This means that there will be measurements which are only half affected by the *Attack(t)* function. These measurements will not have the full effect, and will possibly be less useful in determining whether or not a guard node is a target guard node. If many narrow pulses are generated, many such tainted measurements will be generated. If wider pulses are generated, the ratio between tainted measurements and clean measurements will decrease.

7. Test setup

7.1 Description of the test

We wish to find the best working *Attack* and *Detect* implementations based on the criteria as described in §5.4. To do this, we would ideally perform the attack on the real network by setting up a hidden service and sending attack traffic to it numerous times and compare their results. However, given that the attack will require sending large amounts of data through the network, this is not feasible and ethical due to the stress this will put on the network. As such, the effectiveness of the attack will be calculated based on historical bandwidth statistics of the TOR network. The attack will be simulated by applying the attack function onto the historical data. The amount of attack traffic that a node would have received if it was the target guard node is added to it's published bandwidth statistics. The Detect function will then be applied to the resulting data, and tested for how well it is in distinguishing the simulated target guard node from ordinary guard nodes. Based on the arguments made in §4.5, this method will correspond to the results that are achievable on the real TOR network. All the historical data will be retrieved from CollecTor[13], a project which archives, amongst other things, all the network status documents which contain the consensus score of each node and the published bandwidth statistics of each node. Figure 3 and 4 show the difference between a real-life attack setup and our test setup.

With regards to the influence of the bandwidth consensus score as described in §4.3.3, all the bandwidth statistics will be corrected for the influence of the bandwidth consensus. As such, for each published statistic of a node *g*, we have that:

$$UsedStatistic = PublishedStatistic - 8.12\text{e-}5 * Consensus(g, PublishedStatistic)$$

For U*sedStatistic* the final used statistic by *Detect* and P*ublishedStatistic* the actual published statistic. For the simulated target guard node the simulated statistic with the added attack data will be:

$$SimulatedStatistic = UsedStatistic + AttackInfluence(UsedStatistic)$$

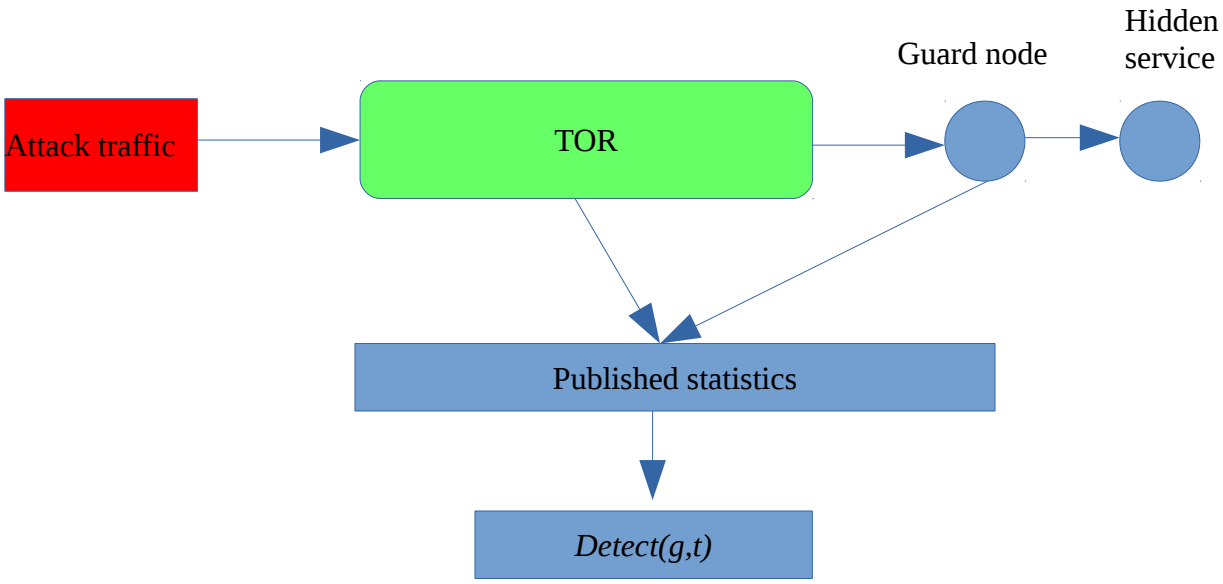

Figure 3: The real-life attack setup

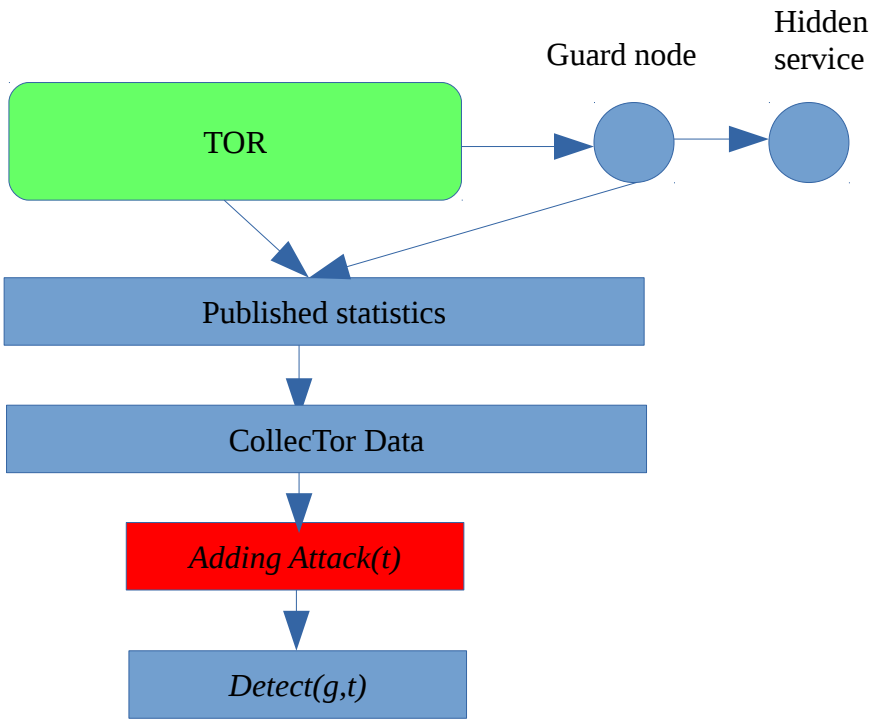

Figure 4: Our test setup

7.2 Type-I and type-II errors

7.2.1 Determining type-I and type-II errors

Since the *Detect* function can err in two ways, either by wrongly marking a node as being the target guard node or wrongly marking the node as not being the target guard node, it is important to measure it's performance in these two dimensions. We will use the statistical convention to refer to these two dimensions as the type-I and type-II errors, assuming that the *Detect* function tests the null-hypothesis that a given guard node is the target guard node. The type-I errors refer to the false rejection of a true null-hypothesis, in this case marking a simulated target guard node as not being the target guard node. The type-II errors refer to the wrong acceptance of a false null-hypothesis, in this case marking an ordinary guard node as the target guard node.

In order to get exact results on the performance of the *Attack* and *Detect* functions, there will be two tests conducted. A time frame will be selected over which the attack will be simulated. The first test will evaluate the *Detect* function for each node on the given time frame without the added attack data, and measure which nodes are marked as the guard node. Since we can assume that no attack with the given *Attack* function has been executed in the past, no node should be marked as target guard node. Each node that is marked is counted as a type-II error.

The second test is conducted on the same time frame, but now with the *Attack* function applied to the statistics of each node. This means that the for each node the bandwidth statistics are updated to what they would have been if they had been the target guard node of an attack during that time frame. By again applying the *Detect* function on each node of the new set, the type-I errors can be measured. Since each node in the new set should be detected, each node that isn't detected counts as a type-I error.

By conducting above mentioned procedures the number of type-I and type-II errors can be determined. However, this only reflects the number of nodes which would be wrongly classified by the *Detect* function, and not the chance that a hidden service uses a wrongly classified node. Recall how the chance that a node is picked as guard node is based on it's consensus weight according to the path selection algorithm described in §4.3. As such, the actual number of type-I and type-II errors are found by correcting each wrongly classified node for its bandwidth consensus weight. This implies that for each type error, not the total number of wrongly classified nodes is produced, but the percentage of bandwidth consensus weight these nodes have in total when compared to the entire network. This means that the final relevant metric will be the percentage of wrongly classified nodes out of the entire network weighted by consensus weight.

7.2.2 Considerations regarding type-I and type-II errors

Now that the procedure to determine the type-I and type-II errors has been established, the next step is to determine which weight should be given to each type of error. A more intuitive way to look at this question is to determine whether it is more important that the *Detect* function marks a lot of nodes from

which it is almost certain that the correct node is among the marked nodes, or that the *Detect* function should mark only a few nodes for which there is a relative high chance that the target is among the marked nodes. An important consideration is the fact that the attack mentioned in §3.2.2 is still viable on individual nodes, and will probably stay viable if the target set of nodes is small enough. By simply DOSing a node, and checking whether the performance of the target Hidden Service is affected, it is possible to determine whether a guard node is the target guard node. This simple attack is not viable on all guard nodes of the TOR network due to the performance requirements of the attacker, but if the set of possible nodes that need to be evaluate is small enough, the attack could provide a confirmation whether or not a suspected guard node is the target guard node. As such, it follows that with regards to the proposed attack that it might be more important to have a somewhat larger set of possible nodes for which there is a high chance that the target guard node is amongst them, than that the set of possible nodes is lower but the chance is also lower. In a practical situation in which the DOS attack is also employed, the rate of type-I errors should be considered more important than the rate of type-II errors.

7.3 Implementation of the *Attack* function

As stated in §6.2, it makes the most sense to create an *Attack* function which alternates. Such an *Attack* function will have the form of a binary wave function. The *Attack* function will alternate between sending zero traffic and sending the maximum allowed traffic, in order to make sure that the difference between affected and unaffected measurements is as large as possible. This implies that while the *Attack* is in its active phase and sending data, it will not use a ramp-up mechanism. Because it is desirable that the measurements that are affected by the *Attack* function differ as much as possible from the measurements that are not affected, it is important to make sure that for every possible guard node, at least one published measurement is available during which *Attack* was in its active phase for the entire length of the measurement. Because the TOR nodes all start their measurements on different times of the day, it is necessary to make sure that the active phase lasts for at leasts the duration of two measurements. Due to the alternating nature of the *Attack* function, it will work cyclic and generate pulses. These pulses can have varying lengths and the *Attack*(*t)* can create a number of pulses during the attack time frame. The amount of pulses that can be generated during a time frame is of course dependent on the length of the attack time frame and the duration of each pulse. During the conducted tests, the *Attack(t)* function will be tested with different amount of pulses, different lengths of the pulses and a different amount of attack traffic send per second during the pulses.

With regards to an actual working implementation of the *Attack* function, the framework of Rochet et al could be used. This application uses the mechanism as described in §4.4 to make sure that the data sent to the Hidden Service is distributed as much as possible across the network to make sure that the only node that receives all the attack data is the target guard node. By doing so, the effect of extra data on the TOR network is minimized.

7.4 Implementation of the *Detect* function

We will test four different *Detect* functions. As described in §6.1.3, the t-test might aid in implementing a *Detect* function. As such, three of the four *Detect* functions will be based on the t-test. The fourth implementation will be much simpler and more intuitive, by being based on the mean. The fourth implementation could be seen as a benchmark for the usefulness of the t-test implementations.

Given the cyclic nature of the *Attack* function as described in §7.3, each pulse of the *Attack* function will create statistics that are affected by the attack traffic and statistics that are not. The detect functions will work by comparing these two groups, and analyzing certain aspects. However, due to the mechanism as described in §6.2.3, there will also be statistics that are partially affected by the attack traffic. These tainted measurements are not well suited for usage by the detect functions, since using them in either of the group would make the groups more alike. Using a statistic during which the attack function was sending data in 25% of the time in the group of unaffected statistics makes this group more similar to the group with statistics which are affected by the attack traffic. It is important to make sure that the groups are as different as possible to increase the effectiveness of the *Detect* function. Figure 5 shows an example of how the measurements are split into two groups in the bandwidth statistics of a fictional node.

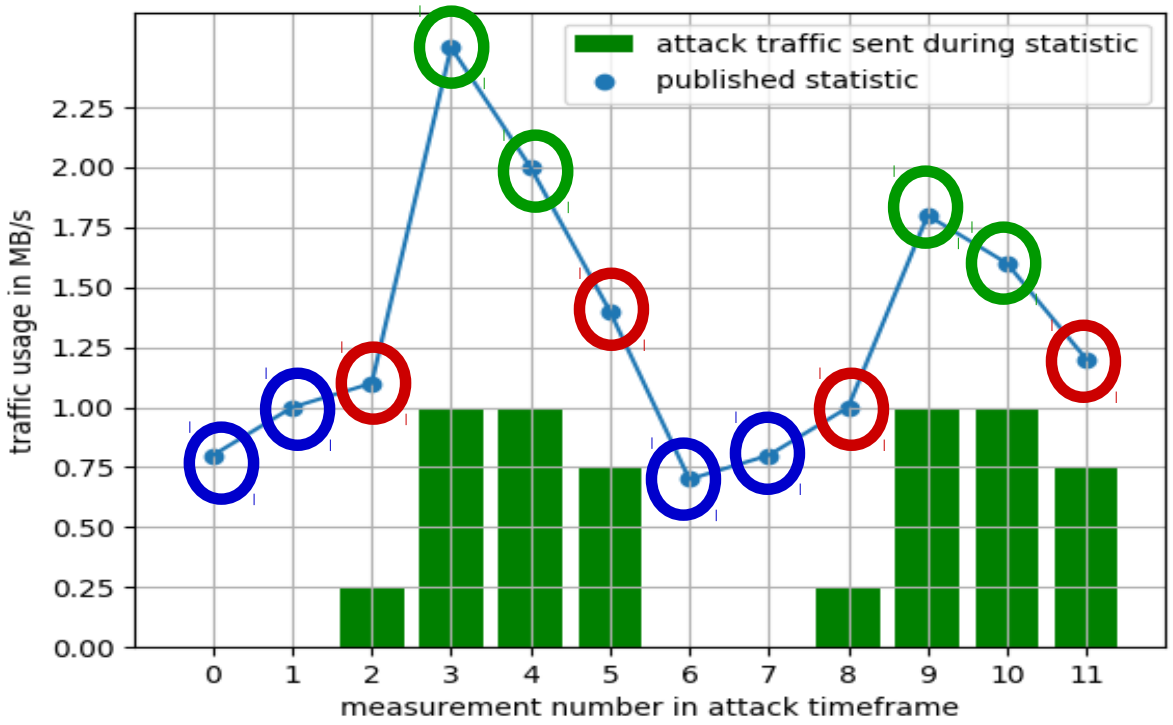

Figure 5: The statistics of a fictional node in a fictional attack

In the figure, all the measurements that are circled blue are statistics which are completely unaffected by the attack traffic. The measurements that are red are tainted as described in §6.2.3, and the measurements that are circled green are maximally affected by the attack data. For the *Detect* function, a group would be created with all unaffected measurements in the attack time frame consisting of all blue circled statistics, and a group of all completely affected measurements consisting of all green circled statistics. All of the below variants will work with these groups. The group with all completely affected measurements will be referred to as *g1*, and the group with all the unaffected statistics will be referred to as *g2*.

To state this more formally according to the model as described in §5, for *t* the attack time frame and guard node *g*:

$$g1 = \{ Traffic(p) \mid p \in Published(g,t) \wedge AttackInfluence(p) = MaxAttack(t) \}$$

$$g2 = \{ Traffic(p) \mid p \in Published(g,t) \wedge AttackInfluence(p) = 0 \}$$

Variant 1: using a the t-test

The first method will be directly based on the t-test. This statistical procedure determines whether two samples who follow the t-distribution are drawn from the same source. The t-test produces a number which becomes higher as the difference between the two samples increases. It is important to note that the two-sample t-test will be used. This requires the t-test to be calculated over two sets. The t-test score will indicate how likely it is that the two sets follow the same distribution. When applied to the problem of identifying the guard node of a hidden service amongst a set of other guard nodes, the t-test will evaluate whether or not the bandwidth statistics which are measured during a time frame in which attack data was sent to the hidden service follow a different distribution than statistics which are measured during a time frame in which no attack data is sent. For a guard node which is not the targeted guard node, the statistics should be unaffected by the data sent through the target guard node, and the t-test should thus indicate that both sets follow the same distribution. For the targeted guard node, the statistics during which attack data was sent should differ from the statistics during which no attack data was sent, and the t-test should indicate that the two sets follow different distributions.

An important note is that the t-test simply produces a metric. No clearly defined cutoff value can be theorized as a threshold for the final detect function. As such, this cutoff value must be determined by experiments as will be done in §8.

The resulting *Detect(g,t)* function is as follows:

First split the measurements collected in the time frame *t* into *g1* and *g2* according to the procedure mentioned earlier. We then have:

$$Detect(g,t) = \begin{cases} true\ if\ ttest(g1,g2) > c \\ false\ otherwise \end{cases}$$

Variant 1a: using the pairwise t-test

Using variant 1 of the *Detect* function, it might be possible change some aspects to achieve better results. By using the pairwise t-test, even more statistical power could be achieved. The pairwise t-test is a variant of the regular two-sided t-test, which assumes that the measurements in the first set are paired to the second set. The pairs are assumed to be dependent on each other. The pairwise t-test has more statistical power if the assumptions about the dependency of the pairs holds. By assuming that measurements which are chronological close to each other are dependent on each other, the pairwise t-test becomes possible. This assumption is also made plausible by breaking down the total bandwidth usage of a node into the guard-bandwidth and the non-guard bandwidth. While the non-guard bandwidth is independent of earlier bandwidth usage, the guard bandwidth changes less frequently, and is thus somewhat dependent on earlier guard bandwidth usage. By paring the measurements which are taken after each other, the statistical power of the t-test could increase.

The resulting *Detect(g,t)* function is as follows:

First split the measurements collected in the attack time frame *t* into *g1* and *g2* according to the procedure mentioned earlier. We then have:

$$Detect(g,t) = \begin{cases} true\ if\ ttestPair(g1, g2) > c \\ false\ otherwise \end{cases}$$

Variant 2: using a double t-test

One shortcoming of variant 1 is that this method does not compare the final results to the expected deviations based on the amount of attack data that was sent. Using a single t-test only indicates whether or not the measured bandwidths both follow the same distribution. If the t-test indicates that a different distribution is used, it doesn't imply anything about which distribution is used. In more statistical terms, using a single t-test could potentially have a large number of type-II errors. The target guard node will always have a high t-test score, but there will be many other nodes which will have a high t-test score for various reasons. To counter this, a second t-test can be used to cut down the number of type-II errors. This t-test will have to determine whether or not the high score on the first t-test was caused by the attack data and not by some other random event.

This will be done by taking the second set of the first t-test, the set with all the bandwidth measurements during the time in which attack data was sent through the target guard node, and subtracting the amount of data that should have passed through the node. For the tests in which the type-I errors are measured, this new set will be transformed back again to the original statistics instead of the statistics with the added attack data. By applying the second t-test to the first set of the first t-test and the newly created set, it is possible to determine whether or not the sets follow the same distribution. If the guard node was the targeted guard node, then the sets will follow the same distribution because the attack data was exactly the data which was added. If the guard node was not the targeted guard node, then subtracting the attack data will create data which follows a totally

different distribution, which is just as likely to be evaluated by the t-test as being from different distributions as the first t-test. This way, the amount of type-II errors will be reduced.

While the adding of a second t-test may seem to be a rather arbitrary choice, it is justified by the fact that the second t-test compares the data to what the data should have been. While the first t-test discriminates between consistent data and non-consistent data, the second data discriminates between non-consistent data which would have been consistent if not for the attack data, and data that would have been non-consistent anyway.

The resulting *Detect(g,t)* is as follows:

Again we split the measurements in the attack time frame *t* into two groups, *g1* and *g2*, according to the procedure mentioned earlier. A third set, *g3*, is created adding each measurement from *g1* to *g3*, and for each measurement subtracting the amount of traffic that was sent by the *Attack* function during the measurement. We have:

$$g3 = \{ Traffic(p) - AttackInfluence(p) \mid p \in g1 \}$$

For a predetermined cutoff ratio *c*, we now have the final *Detect(g,t)* function:

$$Detect(g,t) = \begin{cases} true\ if\ ttest(g1,g2) - ttest(g1,g3) > c \\ false\ otherwise \end{cases}$$

Variant 3: Comparing mean values

One variant which requires less complex statistical tools is comparing the mean values of measurements influenced by the *Attack* function and those that are unaffected. By again dividing the measurements in two groups, with *g1* having the fully affected measurements and *g2* having the unaffected measurements and comparing the means of both sets, it is possible to construct a *Detect* function. This function would be as follows:

By taking *g1* and *g2* as mentioned above, we have:

$$Detect(g,t) = \begin{cases} true\ if\ mean(g1) - mean(g2) > c \\ false\ otherwise \end{cases}$$

7.5 Resulting parameters

Considering the above details of the implementation, the parameters which are available for optimization are: the number of cycles of the *Attack* function, the length of these cycles, the amount of data that is sent during a cycle, the *Detect* variant, and for each individual *Detect* variant the threshold. The number of cycles of the *Attack* function refers to the amount of bursts of data the *Attack* function will produce. It must be noted that this parameter coupled with the length of these cycles determines the total length of the *Attack* time frame. As discussed in §6.2.3 and §7.3, the length of a cycle should be at least the length of four measurements, which is equal to 96 hours in the current TOR network.

Such a cycle will contain one period of 48 hours of not sending any data, and 48 hours of sending the peak amount of data. The amount of cycles that should be produced should then be a minimum of two, due to the fact that the t-tests used by variant 1 and 2 of the *Detect* function require at least two samples per set. As such, the minimum practical attack duration results in eight measurements, which is equal to eight days. The amount of cycles will variate between 2, 3, 4, 6 With cycle lengths of 4 measurements, this will result in attack durations between 8 and 24 measurements. The width of the *Attack* pulse will variate between 2 and 3 measurements. The amount of traffic that will be sent per second during the active phase of the *Attack* function will variate between 0.5, 1, 2 and 4 MB per second. The total amount of traffic that will be sent over the course of the attack will then be between 172GB and 1382GB.

8. Results

8.1 Interpretation of the data

In the data presented below, the parameters mentioned in §7.5 will be tested for various values. The lines in the graphs represent the different variants of the *Detect* function, and each graph will represent different parameters with regards to the *Attack* function. The values that the lines pass through represent the ratio between type-I and type-II errors for the different cutoff values per *Detect* variant. The hypothesis that is tested is the null-hypothesis that the node evaluated by the *Detect* function is not the target guard node. This means that the y-axis, the type-I errors, represent the fraction of nodes that are incorrectly assumed to be the target guard node, and the x-axis, the type-II errors, represent the fraction of nodes that was in fact the target guard node, but wasn't marked by the *Detect* function as such. By varying the cutoff value per *Detect* variant, different ratios can be found. If the cutoff value is too low, many type-I errors will be made, if it is too high, many type-II errors will be made. The fractions are normalized for the average consensus weight per node, and as such, indicate the chance that a TOR client could have selected a node on which a type-I or type-II error has been made by the *Detect* function.

In the data below, the number of cycles made by the *Attack* function is represented by the letter *C*, the amount of traffic sent per second during the active phase of the *Attack* function is represented by the letter *T* in MB per second and the length of a cycle in measurements by the letter *M*.

8.2 Graphs

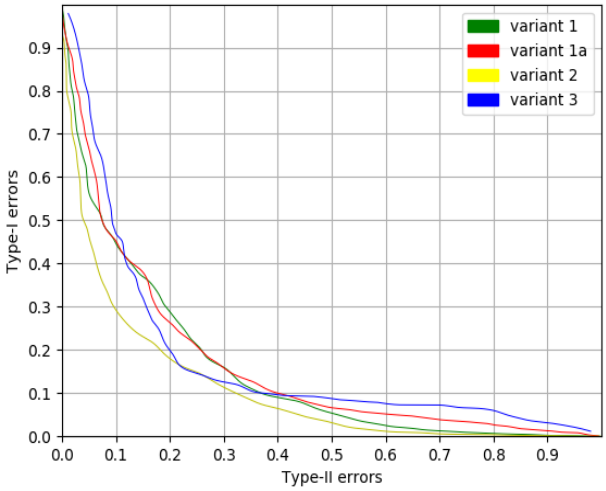

Figure 6: C=2, T=1, M=4

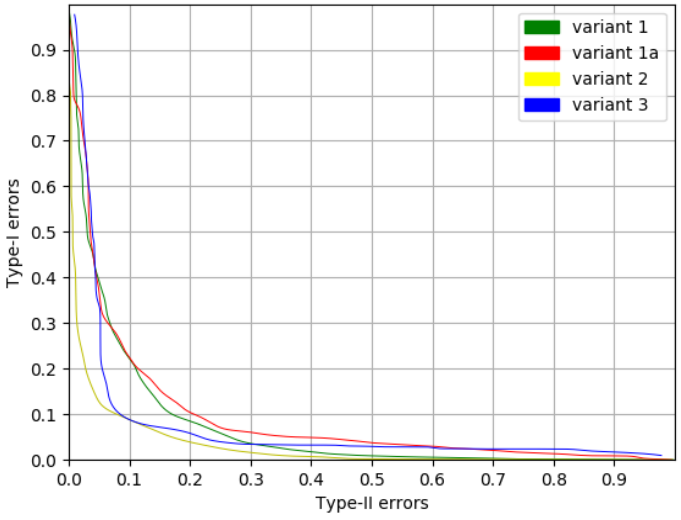

Figure 7: C=2, T=2, M=4

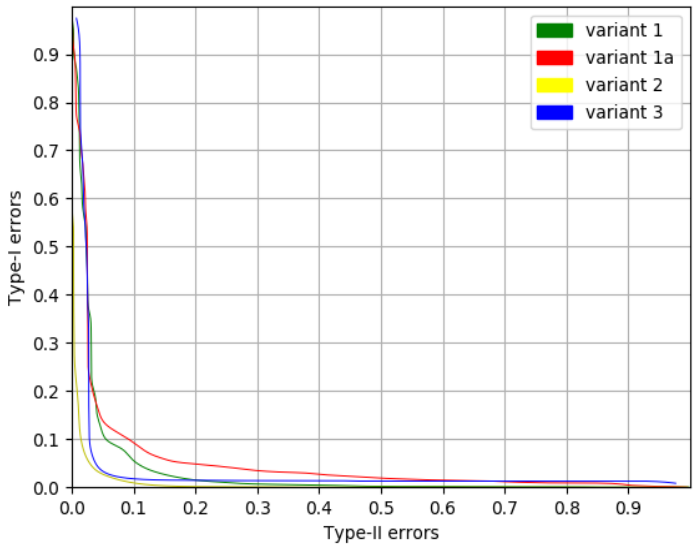

Figure 8: C=2, T=4, W=4

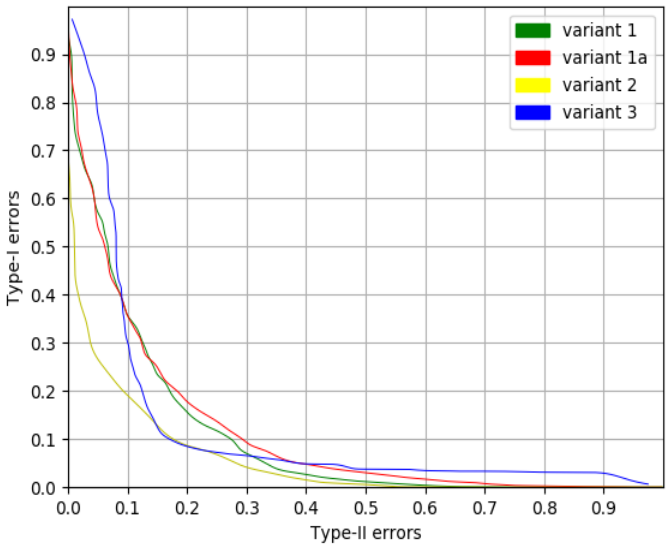

Figure 9: C=4, T=1, M=4

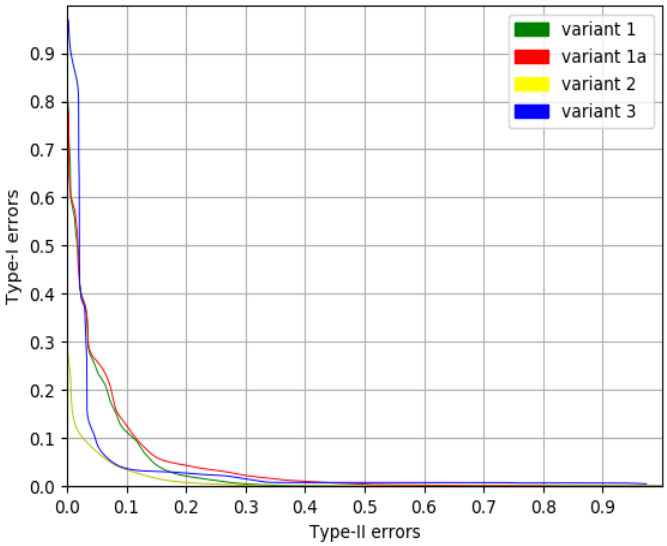

Figure 10: C=4, T=2, W=4

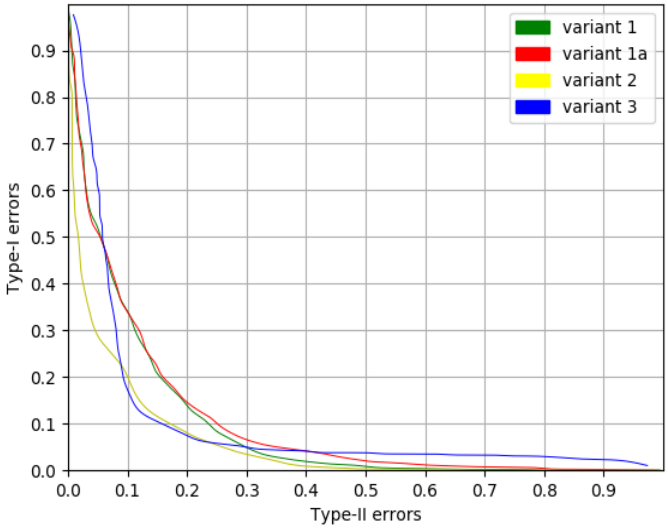

Figure 11: C=6, T=1, M=4

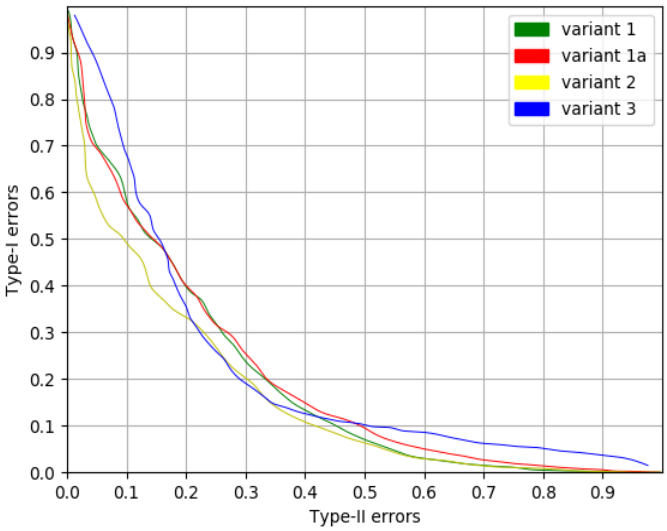

Figure 12: C=6, T=0.5, M=4

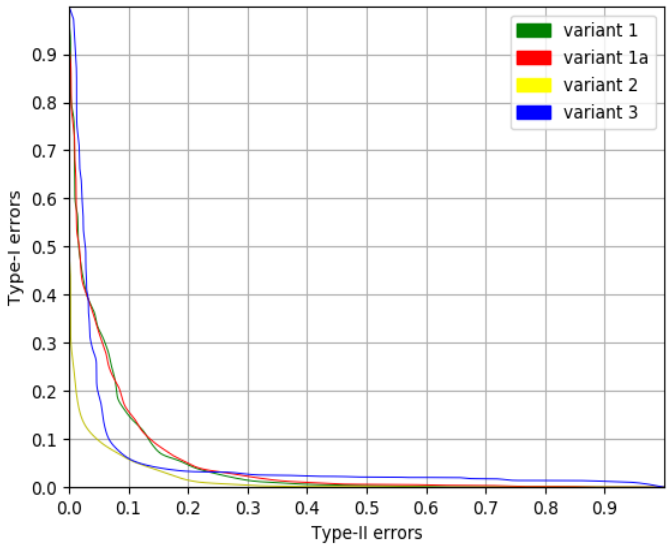

Figure 13: C=3, T=2, M=4

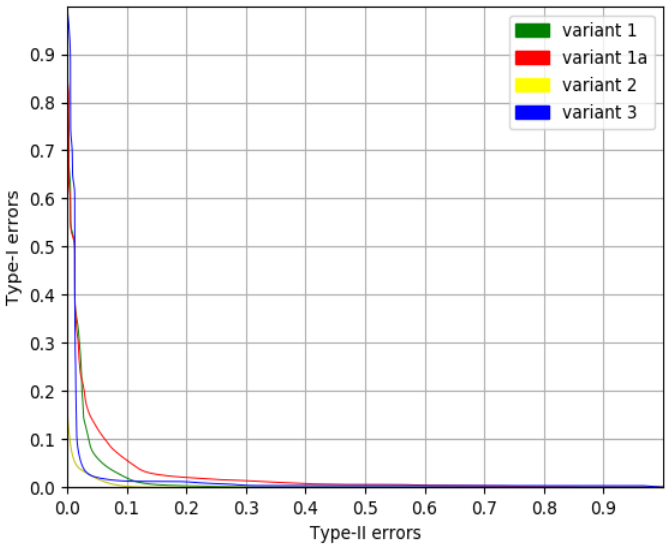

Figure 14: C=3, T=4, M=4

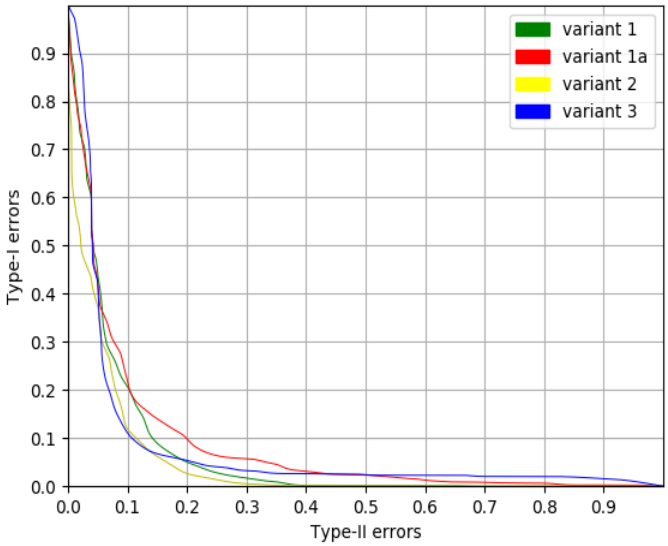

Figure 15: C=2, T=2, M=6

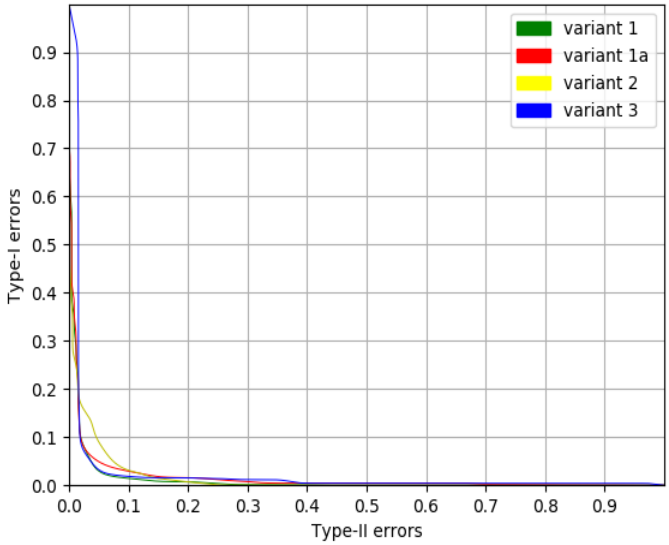

Figure 16: C=2, T=4,M=6

8.3 Tables

As can be seen from the above graph, the most promising *Detect* functions are variant 2 and 3. To compare their effectiveness and determine which *Detect* variant performs the best with which *Attack* parameters, a side by side comparison will be given by comparing the x-values at a given set of y-values and vice versa for the given *Attack* setup. Using these tables, it is possible to determine the type-I and type-II errors when either of the two becomes minimized. Recall that the y-axis corresponds with the type-I errors, and the x-axis with the type-II errors. The tables show you how well variant 2 and 3 perform when certain fixed levels of type-I or type-II errors are used. For example, in table 3a we see that for a type-I rate of 0.1 we have that variant 2 has a type-II error rate of 0.0799. This means that if we accept that 10% of the non-target nodes corrected for their consensus weight are marked as target node, there is a 0.0799 chance that the target node is marked as non-target node.

Y-values	X-values of variant 2	X-values of variant 3
0.1	0.0799	0.0852
0.05	0.1709	0.2182
0.01	0.3471	0.9762
0.001	0.7465	1.0000

Table 3a: C=2, T=2, M=4

X-values	Y-values of variant 2	Y-values of variant 3
0.01	0.4277	0.9752
0.1	0.0880	0.0881
0.25	0.0253	0.0395
0.5	0.0020	0.0294

Table 3b: C=2, T=2, M=4

Y-values	X-values of variant 2	X-values of variant 3
0.1	0.0149	0.0286
0.05	0.0293	0.0386
0.01	0.0965	0.9652
0.001	0.2538	1.0000

Table 4a: C=2, T=4, M=4

X-values	Y-values of variant 2	Y-values of variant 3

0.01	0.1794	0.9590
0.1	0.0088	0.0174
0.25	0.0010	0.0138
0.5	0.0000	0.0127

Table 4b: C=2, T=4, M=4

Y-values	X-values of variant 2	X-values of variant 3
0.1	0.1720	0.1585
0.05	0.2606	0.2987
0.01	0.3903	0.9727
0.001	0.6568	1,0000

Table 5a: C=6, T=1, M=4

Y-values	X-values of variant 2	X-values of variant 3
0.01	0.5997	0.9729
0.1	0.2006	0.1725
0.25	0.0542	0.0578
0.5	0.0030	0.0377

Table 5b: C=6, T=1, M=4

8.4 Insights regarding the *Detect* variants

A first important consideration is that the single t-tests base *Detect* variants seem to perform the worst. They systematically provide more type-I and II errors than variant 2 and 3.

When comparing variant 2 and 3, they seem to perform pretty equal for a large part of the curve. However, as can be seen from table 1a/b and 2a/b, variant 3 seems to systematically under perform when either the type-I or the type-II errors need to be minimized. Variant 3 seems to perform reasonably well when neither of these need to be optimized and only a fair balance needs to be found between the two, but when either of the two needs to be reduced to a lower level, variant 2 seems vastly superior. The explanation that can be given for the phenomena is that variant 3 is only based on the difference between the means of the measurements that are affected by the attack traffic and the measurements that are not. As such, if there is a node that has a large variance in its published statistics, then this might result in a large difference in means. Such a node will only be trumped by a large amount of data added to the actual target node when comparing means. Variant 2 solves this problem by recognizing that such a node has a high variance and assigning it a lower score. As such, variant 3 seems to have problems with the randomness of the data as discussed in §6.1.2.

8.5 Insights regarding the *Attack* variants

Concerning the *Attack* parameters, it is important to note that increasing the amount of traffic send per second has a much larger impact on the performance of the attack than increasing the amount of cycles. While the amount of data transferred is equal between an attack with 4 MB/s and two cycles, and an attack with 2 MB/s and four cycles, the former performs much better. This can be attributed to the fact that the increase in peak value simply makes the measurements stand more out. It can be concluded that it is more effective to have few measurements which are very different than the others rather than having a bunch of measurements which are only slightly different.

Another important thing is that increasing the amount of measurements in a cycle seems to be less efficient than doing more cycles. This can be seen when comparing figure 13 with 15 and 14 with 16. Both of these pairs use the same amount of attack data, both having a total of 6 measurements worth of the active phase of the *Attack* function. As can be seen, the attack setup with 3 cycles and 4 measurements per cycle is more effective than the setup with 2 cycles of 4 measurements. This might appear strange however. Due to the effects described in §6.2.3, the second setup has more usable measurements for the *Detect* methods. However, it appears that the effects of having measurements closer to each other is more important than having more measurements.

8.6 Insights regarding the feasibility of this attack on the TOR network

Given the results above, it appears likely that many Hidden Services could have their guard nodes compromised using any of the above mentioned attack parameters. Even if a Hidden Service only allows for a maximum of 1 MB/s of attack data, by performing six cycles with a width of four measurements it is possible, as shown in figure 11, to reduce the set of possible target guard nodes to two nodes in 35% of the cases.[10] If an Hidden Service allows for a maximum of 4 MB/s, then the same is possible with two cycles of width 4 in 75% of the cases as shown in figure 8. If this attack is used in conjunction with any of the DOS attacks mentioned in §7.2.2 and §3.2.2, then a set of marked guard nodes of twenty would also be acceptable.[11] With the two previous mentioned setups, the possibility of finding the correct guard node grows to 60% and 90% respectively. The guard rotation period is too long to reliably prevent any of these attack setups, since even the longest tested method of six cycles with a width of four measurements only takes 24 days, during which the chance is proximately one in seven that the guard rotation happens.

Even if better performance of the attack is required, it is still possible to perform the attack multiple times in a row. While it is not advisable to attempt to combine the results of multiple attacks if the duration becomes too long with regards to the guard rotation period, the randomness of the bandwidth statistics also implies that it is possible that performing the attack multiple times might result in success in one of the attempts, even if earlier attempts failed.

8.7 The impact of using high-bandwidth nodes

While the above results all make the assumption that a hidden service simply follows the path selection algorithm without any change to the guard node, one exception to this case might be considered. Due to the fact that some hidden services require quite some bandwidth, either due to the application that they run or to be able to withstand DOS attacks, it is not unthinkable that some hidden services only select guard nodes which have certain minimum available bandwidth. The impact of such a policy on our attack is not trivial. On one hand, it is rather obvious that nodes that handle larger amounts of volume require more attack data for the *Detect* function to detect it, but on the other hand, such a policy also decreases the search space. Below are the results of applying the attack setup C=2, T=4, M=4 for different values of the new parameter *B*, which is the minimum average bandwidth usage of a node in MB/s. For example, for *B=10*, only nodes which use over 10 MB/s are considered.

10 The TOR network currently has about 1700 guard active guard nodes, for a type-I rate of 0.001 this would result in approximately two guard nodes being wrongly marked as the target guard node.

11 This would require a type-I error rate of 0.01.

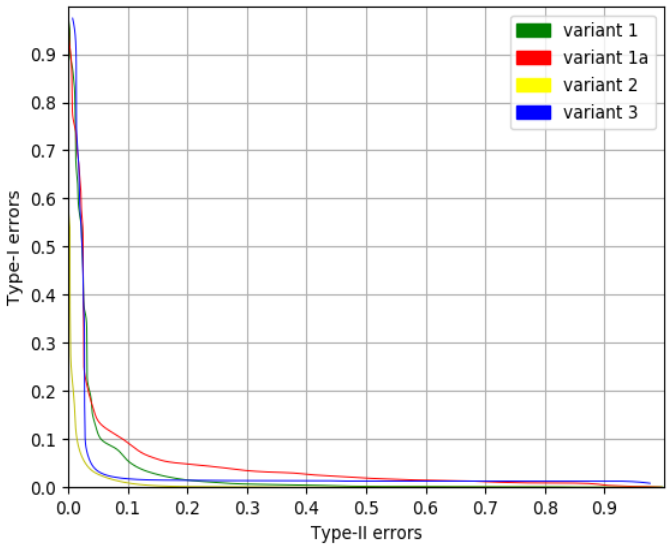

Figure 17: B=0

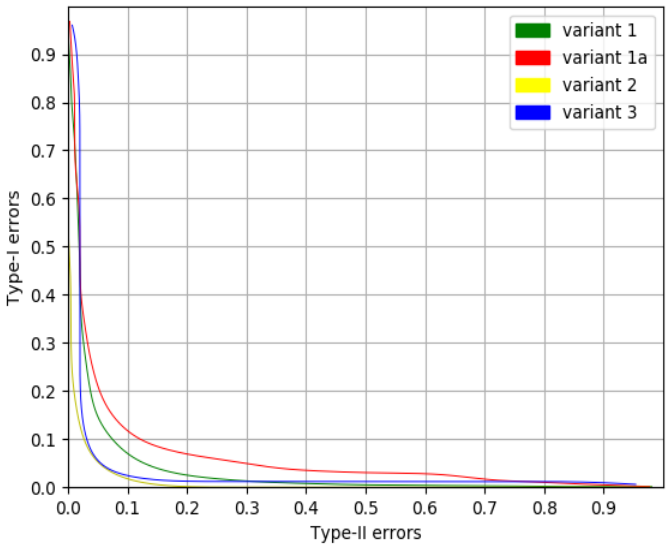

Figure 18: B=5

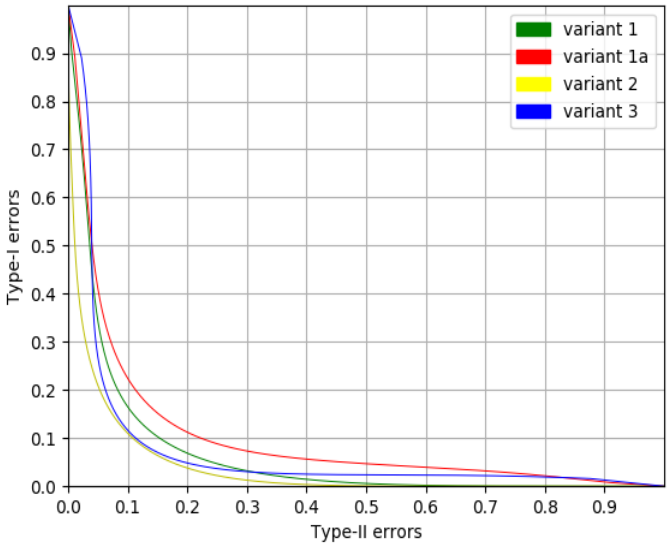

Figure 19: B=10

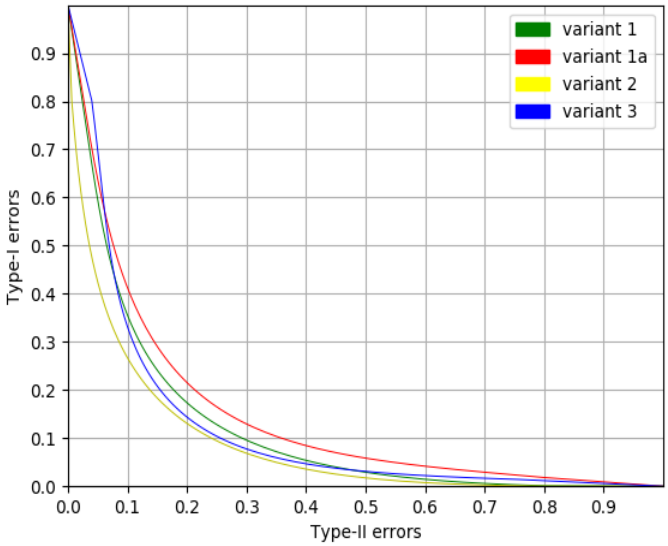

Figure 20: B=15

Minimum Bandwidth	Available guard nodes
0 MB/s	1700
5 MB/s	500
10 MB/s	140
15 MB/s	70

Table 6: available nodes for different values of B

As can be seen, the minimum amount of bandwidth that is required to make sure that the search space larger than 70 nodes is 15 MB/s. If the search space becomes much smaller, the DOS attack starts to become practical again. However, even smaller values of *B* do not hold much practical advantage. While the graphs do indeed show that it is more difficult to determine the correct node amongst nodes with higher bandwidth usage, this is for the most part offset by the reduction of the search space. For example, for *B=10*, only 140 nodes are available. While the x value for y=0.1 is five times as high as the value for *B=0*, a higher y value is acceptable. For B=10 and y=0.05, on average 7 nodes will be selected. Since the x value for y=0.05 is 0.174, the chance of being detected is still 0.826. However, as can be seen from table 4b, for B=0 and x=0.174 the y-value will be somewhere between 0.0080 and 0.0010, which translates to somewhere between 14 and 2 nodes. This goes to show that the effects of using guard nodes that are harder to detect is mostly offset by the reduction in the search space.

9. Possible countermeasures

9.1 Using two guards

As discussed in §2 and §4.6, the number of guards used by a TOR client has been the subject of much debate, and should not be trivially altered. However, it might appear an appropriate countermeasure if it were to solve the problems shown by §8. By using two guards, the attack traffic that is send to a hidden service will be divided amongst these guard nodes. As such, it might appear as this will double the amount of resources required from the attacker to perform the attack. However, matters are slightly more complex than that. While the data is indeed divided amongst the guard nodes, it is important to note that only one of the used guard nodes need to be identified. This implies that while the attack traffic per guard node is halved, the amount of detectable nodes is doubled. While it is not entirely clear how both these factors weight against each other, it is possible that the increase in defense against the attack is very small.

For example, if we were to consider an attack setup with a traffic usage of 4 MB/s, consisting of two cycles and a cycle length of four, then we have that for y=0.01, x=0.0965 if the hidden service uses a single guard node, as can be seen in table 4a and 4b. If the hidden service was to use two guard nodes, then in the same attack setup, each node would receive 2 MB/s. This would mean that for y=0.01, x=0.3471, as shown in table 3a. However, since there are now two target guard nodes that each have a 1 – 0.3471 = 0.6529 chance of being detected, it follows that the chance that at least one of the two being detected equals:

$$1-0.3471^2=0.8795$$

As such, the hidden service that uses two guard nodes has a 1 – 0.8795 = 0.1205 chance of remaining undetected, while a hidden service which uses one guard node has a 0.0965 chance of remaining undetected. Due to the larger number of detectable nodes, the increase in protection that flows from using two guard nodes is negligible, and it doesn’t look like it is a robust way of solving the problem.

9.2 Enforcing bandwidth constraints

Another method of reducing the impacts of the demonstrated effects is by having hidden services limit the amount of bandwidth they will use. A hidden service operator would be required to make an estimate of the amount of legit traffic the hidden service will attract, and limit its bandwidth usage to that expected amount of traffic including some kind of margin. While enforcing a bandwidth constraint will indeed lower the available options for the amount of traffic that can be used for the attack, it should be noted that if more cycles are used, reasonable results can still be produced. See for example figure 11.

Another factor to consider is that this constraint can also not be too tight. If the constraint allows for very little extra traffic, then the hidden service will be an easy target for a DOS attack itself. Considering that the attack might work with amounts of traffic as low as 1 MB/s, enforcing a bandwidth constraint lower than 1 MB/s is required to effectively block this attack. However, enforcing a constraint of 1 MB/s of additional data will make a hidden service a very easy target for a DOS attack.

9.3 Removing the bandwidth statistics

Seeing that it is unlikely that this weakness in the TOR network will be resolved by altering the functioning of the TOR network itself, the only viable option that remains is that the TOR network stops with the bandwidth statistics. At the time of writing, the TOR project has put the removal of the bandwidth statistics on its roadmap to the next major version of TOR, TOR 4.0. The current system will be replaced by a system called ‘Privcount’ which will still produce some statistics about the TOR network without showing the data of the individual nodes[8]. Removing the bandwidth statistics seems to be the only way that this kind of attack can be mitigated once and for all.

10. Future work

10.1 Better *Detect* functions

While the discussed variants of the *Detect* function performed reasonably well and showed that the bandwidth statistics can be leveraged to identify the guard nodes of a hidden service, there are still some untested methods which might prove to be superior. As was noted in the original paper[1], the use of an AI as a classifier might prove to be even more effective. Such an AI would need to train on the expected bandwidth usage of a node and then compare its expected bandwidth usage to the actual bandwidth usage during the attack.

10.2 Consensus score as data source

While the current attack leverages the bandwidth statistics to identify the guard node of a hidden service among a set of 1,700 guard nodes, this source of data regarding individual nodes will be removed. As such, if the attack were to be used even after the introduction of ‘Privcount,’ another source of data should be found in order to still be able to execute a similar attack as the one demonstrated in this thesis. A very interesting and somewhat promising source of data is the consensus score. The consensus score is, as described in §4.3, calculated based on the available bandwidth of a node. If an attacker was to induce target traffic in a guard node, one might expect that the consensus score would drop. One advantage of the consensus score over the bandwidth statistics is that the consensus score is published every hour, instead of every day. And to make the perspective of using the consensus score even better, multiple measurements exists of the same moment since each directory authority publishes a vote which contains an aggregation of its individual received bandwidth authority information. This means that each hour, it is possible to see the measurements of the performance of a node from nine different servers. The only thing that might obstruct the use of the consensus score for a similar attack is the fact that the consensus score is potentially very imprecise. While the published bandwidth statistics of a node are exactly counted and only rounded off on a couple of digits, the bandwidth consensus score is subject to a whole range of external networking effects. If these effects could be characterized, it might be possible to overcome them and use the consensus score as a data source for a similar attack. The impact of such an attack could be much larger than the impact of the current attack due to the fact that unlike the bandwidth statistics, the consensus system is required for the core functionality of TOR and can thus not simply be removed.

11. Conclusion

As was demonstrated in this thesis, the current TOR network is vulnerable to a guard node discovery attack using the bandwidth statistics. We have shown that for a whole range of parameters regarding the attack practical usable results are achievable. The bandwidth statistics which are leveraged to do so are predictable to a small degree by the bandwidth consensus system. The model we applied is very likely closely related to the way the real TOR network works, and because of that the final results are of importance to the real network. In order to achieve the best results, the two-sided t-test method proved to be the most efficient way to identify the target guard node. Using this method, we were able to identify the guard node of a hidden service with very reasonable odds under lots of different attack circumstances. Even if the Hidden Service deviated from the default path selection algorithm and choose guard nodes with a certain minimum capacity, the attack remains viable. It should therefore be considered a good step for the security of the TOR network that the bandwidth statistics are being removed. However, it might be an interesting research to examine whether the bandwidth consensus could be abused the same way as this thesis has shown that the bandwidth statistics can.

Used sources

Literature

[1] *F.* Rochet and O. Pereira, Dropping on the Edge: Flexibility and Traffic Confirmation in Onion Routing Protocols. *In Proceedings on Privacy Enhancing Technologies, pages 27-46, 2018.*

[2] N. Evans, R. Dingledine, and C. Grothoff. A practical congestion attack on Tor using long paths. *In Proceedings of the 18th USENIX Security Symposium, August 2009.*

[3] R. Jansen, F. Tschorsch, A. Johnson and B. Scheuermann, The Sniper Attack: Anonymously Deanonymizing and Disabling the Tor Network. *In 21st Annual Network & Distributed System Security Symposium, 2014.*

[4] T. Elahi, K. Bauer, M. AlSabah, R. Dingledine and I. Goldberg, Changing of the Guards: A Framework for Understanding and Improving Entry Guard Selection in Tor. *In Proceedings of the ACM Conference on Computer and Communications Security, pages 43-53, 2012.*

[5] A. Kwon, M. AlSabah, D. Lazar, M. Dacier and S. Devadas, Circuit Fingerprinting Attacks: Passive Deanonymization of Tor Hidden Services. *In Proceedings of the 24th USENIX Security Symposium, 2015.*

[6] F Dekking, C. Kraaikamp, H. Lopuhaa, L. Meester, A modern introduction to probability and statistics, Springer, 2010.

[7] P. Winter, R. Ensafi, K. Loesing, N. Feamster, Identifying and characterizing Sybils in the Tor network. *In Proceedings of the 25th USENIX Security Symposium, August 2016.*

[8] R. Jansen, A. Johnson, Safely Measuring Tor. *In Proceedings of the 23rd ACM Conference on Computer and Communications Security, October 2016.*

Other sources

[9] Tor rendezvous specifications, https://gitweb.torproject.org/torspec.git/plain/rend-spec-v3.txt.

[10] Tor directory specifications, https://gitweb.torproject.org/torspec.git/plain/dir-spec.txt.

[11] Tor path specification, https://gitweb.torproject.org/torspec.git/tree/path-spec.tx.

[12] Tor Bandwidth Authority specifications, https://gitweb.torproject.org/torflow.git/tree/NetworkScanners/BwAuthority/README.spec.txt.

[13] The CollecTor project, https://metrics.torproject.org/collector.html.

[14] The vanguard addon, https://blog.torproject.org/announcing-vanguards-add-onion-services.