LINKING TOR CIRCUITS

MSc Information Security, 2014

University College London

Otto Huhta

Supervisor: Dr George Danezis

Abstract

Tor is a popular anonymity network, which hides both the content of traffic as well as the identities of the communicating users. We propose a novel attack that reduces the effects of existing safeguards and significantly degrades the anonymity provided by Tor. We show that an adversary is able to successfully link different circuits belonging to the same user by only controlling the middle node on a circuit. This would allow him to observe traffic of a given user for much longer than currently assumed, associate different communication sessions to the same user, and potentially identify the user based on this information.

We record data from our own circuits observed at a Tor middle node and use that data to train a Random Forest classifier to distinguish if two observed circuits belong to the same user or not. Our classifier is able to accurately predict the appropriate class using only information on the timings of circuit control cells and the first and last relay cells on those circuits. We estimate the threat posed by this attack by determining the success probability of an adversary under certain assumptions and discuss potential defences against our attack.

The role of middle nodes on the Tor network has previously received little attention, but we show that it poses a larger threat to the anonymity of users than previously thought. Our findings also raise questions about the security implications of the planned transition to using a single guard node.

Contents

1 INTRODUCTION 7

2 TECHNICAL BACKGROUND 10

2.1 TOR DESIGN AND THREAT MODEL 10

2.2 CIRCUIT CONSTRUCTION 11

2.2.1 Building circuits 12

2.2.2 Predictiveness and circuit rotation 13

2.3 PATH SELECTION 15

2.3.1 Relay information 15

2.3.2 Path length 16

2.3.3 Node selection 17

2.4 RANDOM FORESTS 19

3 ATTACKS ON TOR 21

3.1 PATH SELECTION ATTACKS 22

3.2 TRAFFIC ANALYSIS ATTACKS 23

4 TEST SETUP AND RESULTS 24

4.1 PREPARATIONS 25

4.1.1 Setting up logging 25

4.1.2 Configuration 26

4.1.3 Test Setup 27

4.2 MEASUREMENTS 28

4.2.1 Sample traffic 28

4.2.2 Collected data 28

4.3 CLASSIFICATION 30

4.3.1 Extracting features 30

4.3.2 Random Forest classifier 32

4.3.3 Training the classifiers 32

4.4 RESULTS 33

4.4.1 Statistical results 34

4.4.2 Classification results 35

4.4.3 Distant circuits 40

5 DISCUSSION 42

5.1 RESULT ANALYSIS 42

5.1.1 Statistical results 42

5.1.2 Classifier results 43

5.2 IMPLICATIONS OF RESULTS 45

5.2.1 Base rate fallacy 46

5.2.2 Single guard node 48
5.2.3 Observable bandwidth 49
5.2.4 Extending our attack 50
5.3 SUGGESTED DEFENCES 51
5.4 LIMITATIONS 52
6 CONCLUSIONS 55
6.1 FUTURE WORK 56
ACKNOWLEDGEMENTS 56
APPENDIX: SOURCE CODE MODIFICATIONS 57
REFERENCES 59

1 Introduction

Hiding the identity of a user and preserving privacy online is a challenging task. The Internet was not originally designed with these goals in mind and thus it gives an adversary the possibility of learning who is communicating with whom and possibly even the content of their communications. Today, this is more important than ever with increasing threats ranging from government surveillance to data collection of commercial organizations. Anonymity networks are an important tool for people wanting to protect their privacy and even their life when expressing themselves or simply communicating over the Internet.

A successful example, and the focus of our work, is the Tor anonymity network (Dingledine, 2004), which is designed to conceal the identity and network activity of its users, and frustrate adversaries conducting traffic analysis. Originally, the second generation Onion Router, Tor was designed for the U.S. navy with the goal of protecting government communications in the early 2000's. However, since then Tor has seen an inflation in both traffic and users, and nowadays it is used by millions of people around the world and it consists of over 5000 volunteer relays routing traffic through the network.

Even though all these users rely on Tor for their privacy and safety, the resources for the research and development of Tor are limited. The non-profit Tor Project (TorProject, 2014) consisting of volunteers and a few employees, carries the main responsibility, with contributions from the academic community. However, there are parties, such as intelligence agencies from various countries, interested in identifying Tor users. They have resources at their disposal to discover weaknesses and mount attacks against Tor, while many of the discovered attacks are likely to remain unpublished. The only way to protect Tor users is thus to discover these weaknesses first and publish them in order to devise defences against them. All this highlights the importance of academic research into attacks against Tor.

Research question

The aim of this work is to investigate if it is possible to link different circuits, used to relay traffic in the Tor network, to the same user. More specifically, we are interested in what consistencies are introduced into theses circuits from the way a Tor client builds and uses them. We know that users begin using a new circuit for their traffic at constant intervals, and we wish to learn if this and other information can be used to determine if two circuits belong to the same user.

In addition, we have identified that the majority of previous research neglects the threat posed by a middle node in a Tor circuit. As a result, it is the position that requires the least resources for an adversary to control. This node is still able to observe different user circuits and we thus

want to find out if a middle node actually presents a more significant threat than is generally assumed. Relating to this, we have learned about plans (Dingledine & al., 2014) to move to a single guard node setup, where a Tor client chooses one Tor relay that it uses persistently as the first node on its path to the remote destination through the Tor network. In this discussion the capabilities of a middle node play a very important role.

As a result of these observations, our hypothesis is that the way Tor constructs and uses circuits makes it possible to link together different circuits belonging to the same user by only controlling the middle node of a path. An adversary would thus be able to observe user traffic for longer than deemed safe in the current Tor network. This could enable the adversary to e.g. associate two sensitive communication sessions or identify the user based on usage patterns.

Outline of our work

We begin by presenting the aspects of Tor which have led to our initial observations and which are relevant for understanding our results and the conclusions we draw from these. In Chapter 2, we consider the assumed threat model, and describe how circuits are built and Tor nodes selected for those circuits. The most important design choices in Tor relate to how clients rotate, i.e. build and take up, new circuits around every 10 minutes and how a middle node is selected for each circuit. The concept of guard nodes, and especially plans for adopting a single guard node, is also important. In addition to Tor design aspects, we also present a brief overview of Random Forest classifiers, which we use as an important tool in linking pairs of circuits.

Our work is set into the general context of research on attacking Tor in Chapter 3. We consider a selection of relevant path selection and traffic analysis attacks and highlight how our attack differs from previous research.

In order to test our hypothesis, we needed to gather data on how circuits are created and used in a realistic environment. We constructed a measurement setup consisting of a Tor middle node and a client, described in Chapter 4. The client generated sample traffic using circuits constructed through our middle node which we set up to record timings of certain control cells as well as begin and end times for traffic on that circuit. We describe the necessary configuration settings and source code modifications needed to record the circuit-level data and also reason how and why we avoided recording or releasing any sensitive information related to actual Tor users.

From the collected data, we could then extract features to be used as positive samples when training a classifier to automatically determine if a pair of circuits belonged to the same user. We also created templates from these circuits and used those templates to simulate non-matching circuits scattered randomly in time. Features for negative samples were then extracted from these. We detail the specific Scikit-learn implementation of Random Forest classifiers that we used to classify our samples, the properties of the classifier as well as how we trained it. At the end of

Chapter 4, we provide statistical information on our measured datasets as well as the classification results, which are given for matching circuits separated by different amounts of time. Classifier performance is evaluated both visually using two-dimensional feature spaces as well as observing the true and false positive ratios both directly and via the Receiver Operating Characteristic –curve.

Our results point to strong consistencies in the way circuits are built for a given user, as discussed in Chapter 5. As we expected, there is a strong relationship between circuit create times, so that circuits belonging to the same user are typically created at approximately ten minute intervals. However, the most distinguishing factor was the difference between the time one circuit stops carrying traffic and another one becomes active. This difference is typically very close to zero if the two circuits belong to the same user. Additional features improved the accuracy of our classifier so that we could match two consecutive circuits belonging to the same user with an equal error rate of 8.9%, or achieve a 50% true positive rate with a 0.8% false positive rate.

It is clear that an adversary gains a significant advantage from observing circuit creation and usage. Through subsequent analysis, we show in Chapter 5.2 how in a single guard node setting an adversary would find the next circuit for any given user circuit with a probability of 45%. We also discuss what detection rates can be achieved using a different number of guards, or tuning the classifiers for different accuracies. The importance of the base rate in also considered.

Finally in Chapter 5.3, we consider a few possible defences as well as the limitations of our work. These limitations include the legitimacy of our sample data, the way negative samples were created, the imperfections of our specific Random Forest classifier as well as the general limitations of the applicability and scope of our work. We explain why simply randomizing circuit creation times would not remove the threat and why, in the short-term, the best defence seems to be retaining the three guard node system.

Through our work, we confirm our original hypothesis that an adversary can link two user circuits belonging to the same user with only the circuit-level information available at a middle node. To our knowledge, this is the first study to show that the timings of circuit construction can be used to successfully link circuits belonging to the same user. We also show that a middle node is more powerful than previously thought and that moving to a single guard system could pose unexpected threats to anonymity.

2 Technical Background

In the following, we will go through the important design choices of Tor which enable our attack and introduce the concepts necessary for understanding the reasons behind our decisions as well as the consequences of our findings. We also summarize the theory on Random Forest classification.

We consider the Tor threat model, circuit construction and path selection, beginning with the type of an adversary Tor is designed to protect its users against. We then proceed to describe the circuit construction process, giving details on the control messages used to build circuits one hop at a time, the types of circuits that are built and the circuit rotation Tor uses. Finally we will give information on the way Tor selects the nodes for these circuits. This includes introducing directory servers, reasoning about the optimal path length and describing the procedure of selecting a node for each position individually.

Throughout this work we use the networking terms *client* and *server* to describe the two communicating end points, even though both end points can as well be regular users in the common sense of speaking. We use both *relay* and *node* interchangeably to mean a single Tor Onion Router (OR). Finally, an *adversary* is used to refer to any kind of malicious party with certain capabilities who tries to compromise user anonymity or disrupt service in some way.

2.1 Tor design and threat model

The core of the Tor network is formed by a set of 5000 volunteer network relays, Onion Routers (OR). These routers form an overlay network for routing user traffic and provide directory information about the available relays in the network. Users connect to one of these relays and form a multi-hop path, a circuit, through the network to a desired destination. This destination is either a normal server outside the Tor network or a specific Hidden Service[1], which are services accessible only through the Tor network.

The main goal of Tor is to provide anonymity for its users by making it difficult for adversaries to link multiple communicating parties or multiple communications to each other. The network also needs to provide a sufficient level of performance for interactive applications requiring small delays and adequate bandwidth. To achieve this, for its security Tor relies mainly on routing traffic through the network using unpredictable routes, and the individual relays on the path do not change the characteristics of the traffic they relay.

[1] Hidden Services are services hosted on traditional servers, but which can be accessed only through the Tor network by using a specific procedure for opening a connection without the client actually knowing where the server is located.

This sets Tor, and Onion Routing in general, apart from designs such as Mix networks proposed by Chaum (Chaum, 1981) which also relay traffic through one or multiple relays but where the security comes from batching and altering the order of messages at each relay. Since Tor relays do not alter the traffic in any significant way, traffic can pass through the network with less delay. Unfortunately this also means that the traffic timings remain largely unchanged, which enables an adversary to trivially confirm that the two endpoints he is observing are communicating with each other.

For this reason, while Mix networks provide strong security against a global adversary who can observe traffic at any point in the network, Tor can only protect against a weaker adversary: one who can only have control over a fraction of the network at any one time. This adversary can monitor, modify, insert or drop traffic, run his own network relays or compromise existing relays – but only for a limited set of relays and traffic. This is roughly the threat model[2] employed by Tor. On the one hand this enables the Tor network to employ design choices that increase the performance, i.e. the bandwidth and latency of the network. On the other hand, it can also be argued that this leaves Tor users vulnerable to certain types of attacks and that users cannot blindly assume anonymity in all situations and against all possible adversaries.

2.2 Circuit construction

Tor is not application specific; instead, it supports (and is limited to) TCP based applications. Since TPC is a connection-oriented protocol, the network needs to form stable connections through the network and provide ways of handling TCP streams. This is done by construction circuits through a set of onion routers and by carrying control messages and data along these fixed paths. Different circuits are built for different purposes and rebuilt at certain intervals using different sets of relays in order to frustrate an adversary doing traffic analysis.

When a client wishes to initiate a new TCP connection to a remote host on the public Internet through Tor, the Tor software uses a suitable circuit to establish the connection or builds a new one if none exist. Circuits are constructed on a hop-by-hop basis, first opening a circuit to the first relay on the path and then extending it one node at a time until the final relay is reached. A three-hop circuit is illustrated in Figure 2.1 below. The default path length is currently three nodes: an *entry*, a *middle* and an *exit* node. A Tor circuit is illustrated in Figure 2.1 below. The clients and relays use a single TLS connection (TLS, 2008) between themselves[3] as well as layered public key encryption to hide the contents of messages when routing data within the network.

[2] A threat model describes the resources and capabilities of an adversary.
[3] Multiple circuits traversing that hop are multiplexed into this single TLS connection.

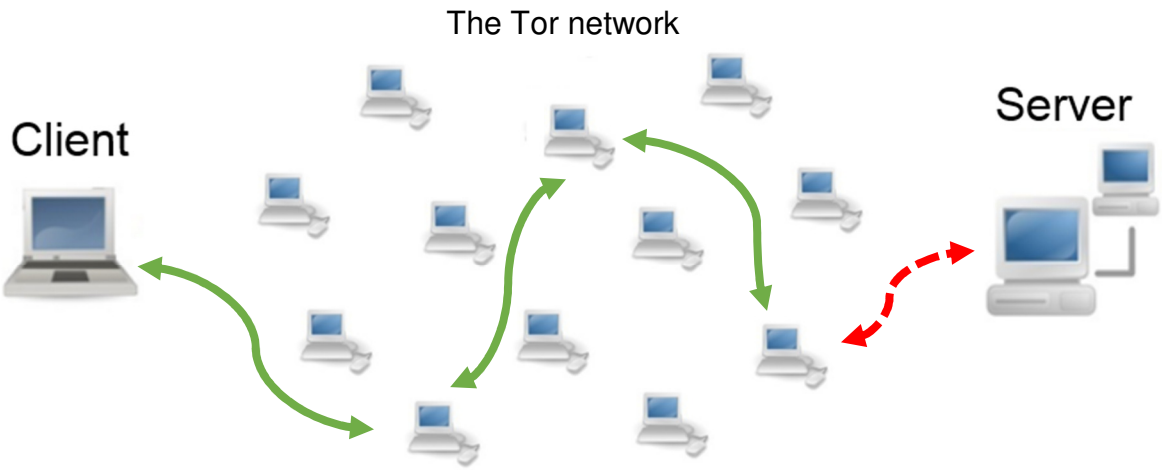

Figure 2.1: A Tor circuit built by a client through the Tor network to a server on the public Internet. Solid lines are circuit hops that use layered encryption and the dashed line represents an unencrypted TCP connection between the exit node and the server.

User traffic is split into 512 byte fixed-sized cells, which are then encrypted multiple times using the public keys of the ORs on the circuit path. More specifically, the client first encrypts the cell using the public key of the last relay on the path, and then with the keys of the subsequent relays in such a way that the final encryption is done with the key of the entry node on the path, the entry node. When this 'Onion' is routed through the network, each relay on the path removes one layer of encryption by decrypting the cell. The final node on the path, the exit node, receives the original unencrypted message. To send data back, the exit node encrypts a cell in the same way but using the public keys in reverse order, beginning with the key of the client.

2.2.1 Building circuits

To open a new circuit, the client uses specific *control* cells to build a single-hop circuit between itself and a chosen OR. There are four different types of control cells: *Create, Created, Destroy* and *Padding cells*. The first three are used for initiating, acknowledging the completion of and tearing down circuits, whereas the fourth one is used for implementing connection keepalive. There is also another type of cell, a *Relay* cell, which is used for actual user data as well as controlling TCP streams. All nodes on a given path can recognize the type of cell from the cell header, whereas the payload is encrypted and only visible to the target OR[4] of each cell.

The client begins by sending a Create cell to the entry node, which in turn replies with a Created cell. These cells are used to exchange key information in order to derive session keys for the new

[4] The client and exit node on a path can observe the unencrypted payloads of relay cells, whereas for control cells only the intended recipient obtains the unencrypted payload.

circuit opened between the two relays. After this exchange, the client now has a functioning one-hop circuit to the first node on the circuit path.

In order to extend the circuit to a second hop, the client needs to instruct the entry node to open a connection to it. This is done with a Relay extend cell containing information about the relay to which the circuit should be extended. The entry node will proceed to opening a new circuit hop to the desired destination, after which it replies to the client with a Relay Extended cell. This same procedure is repeated until the desired path length has been achieved. For tearing down a circuit, the client informs the exit node of this, which promptly sends a Destroy cell to middle node, which repeats this to the entry node, and so on.

Once the circuit has been completed, the client can inform the exit node to open a TCP connection to the desired destination using the Relay begin cell. This connection from the exit node to the remote server is not protected by Tor's layered encryption but instead relies on any encryption the user chooses to use. This makes the compromise of an exit node a significant threat to users, especially as any encryption is dependent on the remote server supporting it.

Based on the previous description we know that the middle node in a circuit will receive and send four control and two relay cells when a client constructs a circuit through it. From the client it receives a Create cell and responds with a Created cell, whereas for opening the next hop on the path, it itself sends a Create cell and waits for a Created cell from the exit node. Between these two exchanges, it has received a Relay Extend cell from the client, which it responds to, after opening the next hop, with a Relay Extended cell. After these it will start observing only relay cells, whose payload it cannot decrypt, until it receives a Destroy cell from the exit node. In addition, typically the first and last relay cells on the circuits are Relay Begin and Relay End cells denoting the opening and closing of some TCP stream(s).

2.2.2 Predictiveness and circuit rotation

A Tor client builds different types of circuits automatically for immediate use or to anticipate upcoming demand, and periodically marks old circuits for close. After marking a circuit for close, it will begin using another circuit for new connections. This design choice is fundamental to our study as our attack is based on how this transition can be used to associate two circuits to each other, and thus the same user.

Predictive circuit construction

Even though multiple TCP streams can be multiplexed over a single circuit, usually one circuit is not sufficient to support all the requests a user might have. Circuits are either *internal* or *exit* circuits, or *fast* and/or *stable*. The former depends on the target address being either a Tor Hidden Service or a server outside the Tor network. We focus only on exit circuits in this study. The latter, on the

other hand, depends on the expected quality of service. Fast circuits use only nodes that have a bandwidth over a certain limit, whereas stable circuits pick only nodes with sufficient uptime.

In principal, Tor prefers using fast circuits for both internal and external traffic, whereas stable circuits are used for certain services ('long-lived' ports). These services, such as FTP or SSH, have long-lived connections and thus require the circuit to remain up for a long time.

Tor clients build fast or stable internal and exit circuits as users request them but it also tries to predict what circuits might be needed in the near-future. There is a slight delay involved in constructing a new circuit before a stream can be opened through it and so whenever there is a circuit ready waiting for a TCP stream, it improves the user experience. There is obviously a trade-off between opening enough circuits to cover all possible user cases and wasting resources opening too many circuits.

At start-up, Tor builds one fast exit circuit for connections to port 80 (used by e.g. HTTP) and two fast internal circuits for hidden services (three if the node itself runs a hidden service). This exit circuit will most likely support other ports as well, but at least it needs to support that one. Once Tor has settled in, it runs an algorithm every second to predict and launch circuits that it expects will be used in the near future. If needed, Tor also builds circuits for performance or reachability testing.

The prediction for building circuits works as follows: for every TCP port used in the past hour, Tor maintains two clean fast exit circuits for regular ports or two stable circuits for long-lived ports, as well as two regular/stable internal circuits for each hidden service used. Not all circuits support all destination addresses or ports, so the client might need to build many pairs of circuits to support all ports the client wants to connect to. A *clean circuit* is one that has had no streams attached to it yet. If the user does not use hidden services, the ones opened up at start-up will slowly die out, and no new ones are built until a user requests one. There is also an upper limit of 12[5] open clean circuits after which Tor will cease building circuits predictively.

Circuit rotation

Users rotate to using new circuits approximately every ten minutes to limit the time a node on the path can observe the traffic. Using the same circuit for the whole session duration might give the adversary enough information for identifying the user or linking together two different streams of that one user. On the other hand, constantly building new circuits wastes resources (both at the user and at the network level). Previously circuits were rotated at a twice-per-minute basis, but nowadays Tor has settled for a compromise of 10 minutes.

[5] 12 according to the specification, but 14 seems to be the value used in the source code.

This does not mean that the lifetime of all Tor circuits is exactly ten minutes, but that at that moment Tor stops using a specific circuit for *new connections*. A circuit remains open for any long lived connections since, as TPC is a connection-oriented protocol, one cannot easily switch all active streams to a new circuit (and exit relay). When a stream is attached to a clean circuit, the circuit in question is marked *dirty*. Tor goes through circuits periodically and after ten minutes marks a dirty circuit for close. Once there are no longer any active (open) TCP streams on this circuit, it will be torn down. Overall, while we expect the lifetime of a circuit in general to be close to ten minutes, they can actually remain open for an arbitrarily long period of time.

This increases the risk of using services that connect to the previously mentioned long-lived ports. Since these services maintain the connection open for a long time, they will also use the same circuit for the whole duration of the session. This means that a malicious node on the chosen circuit path can observe traffic for this user for an extended period of time.

2.3 Path selection

The security of Tor, and onion routing in general, comes from the assumption that it is difficult for an adversary to monitor all the nodes on a given path. As stated in the threat model, it is assumed that an adversary can control a fraction of the nodes and traffic on the network and thus some user circuits will inevitably contain compromised nodes. Selecting paths must then, on the one hand, minimize the probability of choosing compromised nodes and on the other hand, ensure that security holds even if a user picks a malicious relay as part of a circuit.

We consider here three design choices that affect the security of the Tor path selection. Firstly, we consider what information the clients base their node selection on; secondly, what a sufficient path length is and finally, how exactly the nodes for this path are picked.

2.3.1 Relay information

The relays in the Tor network are operated by volunteers around the world and as such there has to be a way for users to reliably learn which relays they can use to construct their circuits. This is a common issue in networked systems, and Tor solves it by using a small set of *Directory authorities,* which receive information from the onion relays, determine a common view of the current network, and give out this information to clients requesting it.

Every now and then each OR in the network creates a signed *router descriptor* and uploads this to a directory authority. This descriptor contains the properties of that relay as well as information necessary for others to open connections to it. Directory authorities serve these descriptors to clients with the help of normal ORs acting as *Directory caches*. Periodically they also form a complete view, a signed consensus, of all the currently available descriptors (i.e. relays).

Clients, directory caches, as well as the directory authorities themselves use this consensus to determine which old relay descriptors they should discard and which new ones to download.

The directory authorities also have the important task of assigning status flags to relays. These flags give general information on the role or capability of that relay. For example, the fast and stable circuits described in section 2.2.2 are built using relays with "Fast" and "Stable" flags in their descriptors. Others possible flags are the *Guard (entry), Exit* and *BadExit* flags determining eligibility for a specific position in a circuit, the *Named, Unnamed, Valid* and *Running* flags determine if the router is named and functioning properly and *Authority, HSDir* and *V2Dir* flags signify an authority server or a directory cache.

Relays cannot directly choose these flags; instead, they need to influence the decision that grants them a specific flag. To prevent a relay from manipulating e.g. his uptime or bandwidth figures in order to receive a flag or gain an unsubstantiated share of the network traffic (Bauer & al., 2007), Tor employs active measurements of running relays. More specifically, the availability and bandwidth of each relay is periodically measured and this *measured* value is the one used for making flag decisions at authorities or when clients choose appropriate nodes for path construction. More information on the flags and the directory servers can be found in the Tor Directory protocol (Tor, 2014).

2.3.2 Path length

An important issue that has been debated extensively is what path length should be used for circuits. A short path length makes it possible for just one malicious relay to compromise anonymity, whereas a long path length carries a performance penalty with it. When considering connections originating and terminating in the public internet, the two weakest points in a circuit are the entry and exit nodes. Here the weakest means that those nodes learn the most information about the two communicating end-points.

In case of the entry node, it can directly learn the address of the client, whereas the exit node discovers the remote end-point of the connection and is also capable of observing the unencrypted[6] traffic leaving the Tor network. This does not apply to the entry node since traffic is encrypted between the client and the first hop. It is clear then that these two positions should not be assigned for the same relay, as is the case with a normal proxy, where the proxy sees both incoming and outgoing connections. Using a two-hop path is a better solution but still an adversary controlling one node on the path trivially learns of the other relay on the path, i.e. either the entry

[6] That is, not encrypted by the Tor layered encryption. Any encryption (such as TLS) is dependent on the server supporting it and the client enforcing it.

node learns who the exit node is or vice versa. As a result the adversary can target this attack in order to either compromise it or take it offline.

What Tor has opted for is a compromise of a default three hop path length. This way there is an additional node obfuscating the link between the entry and the exit nodes and compromising either of these nodes does not directly give the adversary knowledge of the other end. Also, as it is between two onion routers in the circuit, the middle node can only observe encrypted traffic and it cannot directly deduce the identity of the user or the remote end-point.

One might think that increasing the path length further would at the same time increase security. Unfortunately, in addition to the performance penalty, it is not clear if it actually enhances anonymity. The more nodes are involved in transporting a circuit, the more times the same cells are relayed in the network. As to security, the probability of selecting a malicious node for a specific position in the circuit remains the same, but the probability of selecting such a node for *some* position in the circuit actually goes up as the path length is increased. Thus considering this, it seems that a path length of three strikes a good balance between good enough security and a tolerable performance penalty.

2.3.3 Node selection

Besides the path length, the selection of nodes needs to be carefully considered to minimize the possibility of selecting multiple malicious nodes within one circuit path. In general, the nodes are selected individually for each position following two main principles: first determining which nodes are eligible for that specific position and then selecting randomly from that set. The selection is not completely random, since the nodes are first weighted according to their measured bandwidth. This is done to distribute the load more evenly among the network relays; higher capacity relays are included in more circuits and thus end up carrying more traffic.

When forming the list of eligible nodes, the clients rely on the flags assigned to each relay by the directory authorities. Entry nodes require a Guard flag, exit nodes require an Exit flag and must also not have a BadExit flag, whereas both need to have Valid and Running flags. The only flag needed to be eligible for a middle node position, however, is a Running flag. That is, as longs as the relay has been online for the past 45 minutes, it can be chosen as the middle node. In addition to flags, the client uses a few other rules to filter out relays from the list of possibilities, such as avoiding using relays that belong to the same network address family.

Exit node selection

The most defining feature of a circuit is what services (ports) it can support and thus the first node selected for a path is the exit node. From the set of valid and running exit nodes, a Tor client picks out those that support the type of connection the user is requesting. More specifically, it looks at

the requested remote IP address and port and determines which relays allow connections to these destinations. Each compatible relay is then weighted based on its bandwidth capacity and the selection is made randomly from this set.

An important observation here is that not all relays support exit traffic. Each relay operator can define an advertised *ExitPolicy*, i.e. what remote IP addresses and ports it is willing to open connections to. The reason for this is that, while it is not illegal to run a Tor exit relay, running one can still result in abuse or legal threats since the address of this exit node is the one seen by remote endpoints outside the Tor network. While relay owners might want to avoid supporting certain abuse-prone ports (such as port 80), they can still allow exit traffic for other services with their Exit policies. This way they can contribute their bandwidth to the Tor network. If a relay prohibits all exit traffic, it is simply used for other positions in the circuit.

Entry Guards

After the exit node, the Tor client proceeds to selecting the first hop on the path. For this position, Tor implements a specific mechanism called *Entry Guards*. Each client maintains an ordered list of three nodes, called entry guards, from which it randomly chooses one to be used as the first hop. A guard remains on the list for 4-8 weeks and new nodes are added to the list only if the ones already on the list become unusable. This can happen if a node loses one of the required flags or goes offline. However, the original node is not removed from the list, and if it later becomes usable again the client reverts back to using it.

For a node to be considered as an entry node, it must have obtained a guard flag which in turn is awarded based on three metrics: bandwidth capacity, uptime and time the node has been known. For example, currently a new node can only achieve the flag once it has been in existence for at least seven days (it receives the flag on the eighth day) and has at least a 2 MB/s bandwidth. The next chapter (Chapter 3) briefly touches upon the attacks that have led to the introduction of entry guards and why there are plans to move to using a single guard node.

Middle node

After determining the entry and exit points in the network, the client fills in the gap between them by choosing a node for the middle position(s). As previously stated, in general all that is needed is for the node to be running. There are no minimum bandwidth limits, but nodes are again chosen with a probability proportionate to their bandwidth capacity.

With the default path length of three, the client obviously chooses only one middle node, but in case the user wishes to construct a longer circuit, the same algorithm is applied to each extra node between the entry and exit nodes. Even though middle nodes allow no traffic to exit the Tor network from their relay, they can be used as exit nodes for connections into hidden services. This

is because, unlike in the case of normal exit traffic, the exit traffic to a hidden service is encrypted in a similar way as the traffic between two ORs.

2.4 Random Forests

Random Forests (Breiman, 2001) are an ensemble machine learning method, which can be used to analyse large and complex datasets. They can be used for different kinds of tasks, of which the most popular ones are *classification* and *regression*. In the first one, the aim is to be able to recognize the type or category of a given sample, such as if a medical test result indicates the presence of a certain disease. In the latter case, we aim to predict a value based on known samples, such as predicting the price of a stock as a function of the financial performance of the company.

Our goal is to determine from recorded circuits, if two of those belong to the same user. This is a classification task which we aim to solve by using a Random Forest classifier. For that reason we give a short introduction to the theory behind Random Forests and the important properties affecting their performance. We also define various metrics for describing the performance of a classifier, as well as introduce some ways of visually illustrating it.

The idea behind ensemble methods is to use a collection of so called weak learners to improve performance over single-learner methods. In the case of Random Forests, these weak learners are individual *Decision Trees*. Decision trees are a hierarchical structure of split and leaf nodes, where each split node represents a 'question' posed to an individual sample, whereas each leaf node corresponds to a label given to that sample. Samples can be though to be inserted into the tree at the top and successive questions are asked at each split node with the next question always depending on the 'answer' to the previous question. Once a sample reaches a leaf node, it is labelled with the label associated with that leaf node.

To determine the appropriate leaf nodes and the questions to ask at each split node, the tree needs to be *trained*. When used for classification, this is done using set data points whose label is known in advance. Each data point, or sample, consists of *features*, i.e. any underlying properties such as a pixel in an image or packet volumes in traffic analysis. A tree is trained so that each node optimally splits the samples using some function applied to the features of the samples where the optimality of a split depends on the specific setting. For classification the tree might e.g. use axis-aligned functions to maximize the *information gain* at each split node.

For information gain, an optimal situation with binary data would be one where the sample set is split into two pure subsets both containing only samples with one of the labels. However, this is often not the case and so each successive node keeps splitting the samples into even smaller partitions resulting in more and more braches. Thus it is typical to limit the growth of trees by

determining a maximum depth or the smallest possible subset of samples at which point we stop growing all branches. Once we stop growing the trees, each leaf node is assigned a label based on the known labels of the samples that have ended up at that leaf. After a tree is trained, these labels and the individual split functions for each split node are stored for testing.

Tree *testing* is the process of inputting an unseen[7] sample into the tree and getting the predicted label as output. Each split node uses the previously determined split function to forward the sample to either the left or the right child. The label of the sample is determined by the final leaf node it reaches. Naturally the more accurately a sample ends up at a leaf node with a correct label, the better the tree is.

A Random Forest uses multiple trees to determine the final label of a sample. Each tree is trained individually and the outputs are combined to give one single prediction through e.g. averaging. A subset of all the samples is selected at random for each tree so that each tree is built using different data. Moreover, combining this with the fact that each split function is tuned using a random subset of the available features, results in trees that each classifies the data in a (mostly) different way. The key aspect here is that each tree is randomly different from each other, leading to low correlation between different trees which results in better generalization and robustness.

The most important properties affecting the accuracy of a Random Forest are the forest size, the type of weak learner used[8], the maximum allowed tree depth, amount of randomness in tree construction, choice of energy model. Also, the choice of features plays an important part in practical applications. (Criminisi, et al., 2012) Our description of Random Forests here is necessarily brief and more detailed information can be found in (Breiman, 2001).

Performance metrics

Finally, we give a short introduction on how we evaluate the performance of our classifier. When it comes to the outputs of a classifier, we can have four different situations based on the prior condition, and the result of the classification. In our case, the prior condition would be that a pair of circuits either belong to the same user or they do not. In both cases our classifier may, however, label the pair as either of these, depending on how it has been trained.

It is clear that two of these possible outputs would be wrong, and two would be correct. If the prior condition is positive, i.e. that we have a match, our classifier produce either a t*rue positive* or a *false negative*. As the terms imply, these mean that either the sample is classified correctly as positive, or falsely as negative. In the negative prior situation, the results are correspondingly either *false positive* or *true negative*. In our study we look at the probabilities of these events, or *rates*, among

[7] A sample not used in the training phase.
[8] That is, the class of decision function, such as hyperplanes or other surfaces of higher degree of freedom.

all the outputs of our classifier. The number of a certain type of situation is divided by the total number of possible situations. For example, the true positive rate is computed by dividing the number of trues positives with the sum of true positives and false negatives, the false positive rate is the division of the false positives with the sum of false positives and true negatives, and so on.

To use these rates to describe the performance of a classifier, we can use a popular method called the Receiver Operating Characteristic curve (ROC-curve). A ROC curve plots the true and false positive ratios for different decision thresholds. The computation begins with a high decision threshold which results in a zero rate for both. This threshold is then gradually lowered to observe how the true and false positive rates increase. This figure is useful, since it gives information on the performance of the classifier for multiple different scenarios.

Two general performance metrics can be derived from the ROC-curve, or more precisely, based on the different true and false positive rates. The Area Under the Curve (AUC) metric literally gives the area under a ROC-curve, and can be used to compare the general performance of two classifiers with different data in the same setting. Another metric that can be used for this is the Equal Error Rate (EER) which is, as the name describes, a situation when we have an equal rate of false positives and false negatives. Due to the relationship between these different ratios, the EER can be computed as (1 – true positive rate), and it can also be extracted from a ROC-curve. In general, the higher the AUC and the lower the EER, the better the classifier is. The ROC curve and these values are visualized in the Results (Chapter 4.4).

3 Attacks on Tor

Over the years, various types of attacks have been proposed against Tor. There are also some attacks which are outside the previously defined Tor threat model and against which Tor does not aim to protect its users in the first place. Such attacks are for example *traffic confirmation attacks*, where an adversary is capable of observing traffic at both communicating endpoints. Since Tor relays do not alter traffic in any significant way, the adversary is able to trivially confirm that the two parties are communicating with each other by simply observing volumes and timings of traffic. Additional attacks outside the threat model include application-level attacks where the user is deceived e.g. through malware into revealing his identity outside the Tor network or after he has stopped using Tor.

We consider here attacks related to path selection and traffic analysis. The first are relevant in influencing the path selection process and respective attacks have led to the development of active bandwidth measurements and the guard node system. Our attack, on the other hand, is part of the large group of traffic analysis attacks. We consider a few general ones and a few more specific ones related to linking different streams to the same user, before detailing how our own attack differs from the existing research.

3.1 Path selection attacks

Tor relies on picking unpredictable routes through the network and for this reason influencing this process is an attractive target for attack. The ultimate goal for an adversary is to control one or more nodes in a given circuit. He may try to either influence the probability that his node is selected or wait until he eventually becomes part of the target circuit. Bauer et al. (Bauer & al., 2007) show that by falsifying the advertised bandwidth capacity of its relays, an adversary can successfully replace all entry nodes with his relays. As a result, active bandwidth measurements are now performed on the relays to acquire a more truthful bandwidth value.

In the predecessor attack (Wright & al., 2004), the adversary uses the fact that if nodes for circuits are chosen at random, over time the client will eventually include a given (malicious) relay into one of his circuits. Borisov et al. (Borisov & al., 2007) show that the required time can be shortened by forcing the client to build new circuits by using selective denial of service attack. They propose that an adversary who controls a set of Tor relays repeatedly terminates circuits the client builds through these nodes, forcing the client to build new circuits. These DoS attacks are continued until the adversary controls both the entry and exit node of a circuit and can identify both the user and the target server.

Wright et al. (Wright & al., 2003) propose to fix the entry and exit nodes to defend against the predecessor attacks, whereas Øverlier and Syverson (Overlier & Syverson, 2006) expanded on this by proposing the use of a fixed set of entry guards. As a result, Tor now employs three guard nodes, even though Borisov et al. (Borisov & al., 2007) argue that this makes the situation worse for those users who pick malicious entry guards and that the use of three guard nodes leads to the largest amount of compromised circuits.

These three guard nodes also work as a fingerprint for that user. The reason is that the probability of two different users picking exactly the same three nodes (at random) is very small, and thus if an adversary can learn these guard nodes for a given user, he can accurately identify connections belonging to this user. This has in turn resulted in the current plans for moving to a single guard node design (Dingledine & al., 2014).

3.2 Traffic analysis attacks

Traffic analysis uses traffic metadata, such as traffic volumes or timings, to infer information about traffic contents or the communicating end points. These are fairly common attacks since they are (usually) within the Tor threat model, and thus potentially reveal some unanticipated threats or weaknesses. We briefly consider a few representative attacks and then proceed on to detailing our own attack.

In a recent analysis, Johnson et al. (Johnson & al., 2013) show that Tor users are susceptible to traffic correlation by malicious relays that eventually control both entry and exit nodes, as well as by network-level adversaries who can observe large portions of the Internet. It seems that peer-to-peer applications and users who use long-lived ports are especially vulnerable. Ling et al. (Ling & al., 2009) propose an attack based on cell counting, where the adversary who controls both the entry and exit nodes of a circuit embeds a detectable signal into the traffic stream at the entry node by varying the number of cells flushed at the entry node. The resulting variation can be observed at the exit node to confirm presence of target traffic.

A seminal work by Murdoch and Danezis in 2005 (Murdoch & Danezis, 2005) shows that an adversary can discover the nodes on a circuit path by building probe circuits through different Tor relays to measure possible delays introduced by traffic originating from an adversary controlled server. The server introduces a pattern into the client traffic, which in turn affects the latency experienced by other circuits using a given node on the path. However, four years later Evans et al. (Evans, et al., 2009) conclude that this attack would no longer be feasible with the size of the Tor network at the time. Today, the network is even larger than in 2009.

Related to the previous attack, there are also so called website fingerprinting attacks, where the adversary builds a fingerprint for known websites and looks for that fingerprint among user traffic. A study by Wang and Goldberg (Wang & Goldberg, 2013) shows that an adversary controlling an entry node (or the link to the client) can identify with a very high accuracy which website, out of 1000, a user visited. Their work builds on previous work by (Panchenko & al., 2011) and (Cai & al., 2012).

Two additional attacks related to linking different streams to the same user edge slightly closer to our proposed attack. Blond et al. (Blond & al., 2011) show that an adversary can discover the identity of a user by taking advantage of the simultaneous use of a secure and an insecure application. The real address of the user is leaked through the unsafe application, which then allows the adversary to track additional (secure application) streams belonging to the same user. In the attack proposed by Hopper et al. (Hopper, et al., 2010) the adversary measures the round trip time

for a client connecting to two different adversary controlled servers and uses this to associate two different streams to the same circuit, i.e. to the same user.

Our attack

The previous selection of attacks highlights the general trends of attack research on Tor. The goal is often to control either the entry or exit nodes, or both to identify the client and remote server. In addition, the adversary can conduct traffic analysis on the timings and volumes of data, possibly using fingerprinting or introducing some patterns into the traffic himself. Our work deviates from this general trend.

We propose a traffic analysis attack where an adversary controlling a *middle node* observers circuits created through it and is able to link these circuits belonging to the same original user. An adversary is not directly able to discover the identity of either endpoint but is able to profile the traffic for an extended period of time and significantly reduce the anonymity provided by Tor. Our attack is the first of its kind, and our study is in the minority of studies considering an adversary controlling only the middle node in a circuit.

4 Test Setup and Results

In order to test our hypothesis, it was essential to gather information on the behaviour of real circuits in the Tor network. To achieve this, we ran a middle relay modified to record circuit events during a period of time in July and August 2014. The goal was to gather enough information to discover patterns between different circuits belonging to the same user.

We realize that there are serious privacy concerns associated with collecting data from an anonymity network, both with what is collected and how. We carefully evaluated the possible privacy issues and went into great lengths to ensure our measurements caused no disruption of service and that the statistics collected or presented in this work cannot be used to identify any users. To that end, this report contains no addresses, names or dates for any collected data.

Our main goal was to build and record our own circuits without the risk of recording anyone else's traffic. To achieve this, we originally set out to using a private middle node whose identity is not known to anyone else and more importantly is not included in the consensus used for path selection. Unfortunately we encountered unexpected problems: our node would not relay traffic as a middle node if it was not included in the consensus. Circuit construction seemed to proceed as expected, but no streams could successfully be attached to it.

We could not pinpoint the reason for this behaviour, especially since additional tests showed no such problems if our node was used for the exit position or if we used any publicly available node for the middle node. This included our own node once it was made public. We contacted some Tor developers for advice, but even they found no explanation for this behaviour. In any case, this eventually forced us to make our relay public in order to gather any data.

Our measurement setup consisted of a local Tor client and an externally hosted Tor relay. We reasoned that since each client builds the circuits according to the same specification, we would get representative data even with a single client. This client was modified to create all its circuits using our relay as a middle node, log circuit information and use these circuits for HTTP traffic. The relay in turn was modified to log observed circuits and their traffic. Features were extracted from the gathered measurements and used to train and test a Random Forest classifier to get results on how well two individual circuits can be linked to their user.

This chapter details the measurements carried out, including the required preparations and the subsequent classification, as well as presents the results of these measurements. The preparations included carefully choosing the data points to be collected and modifying the source code to record these. Details are given on the actual physical setup and what data was gathered in the measurements phase. We also lay out the classification process: what features were extracted and used, why we chose a Random Forest classifier and how the classifier was trained and used to conduct the classification. Finally we present the actual results of the measurements, which will be analysed in the following chapter (Chapter 5).

4.1 Preparations

Before the actual measurements could be carried out, the client and relay needed to be configured, and certain modifications to the Tor source code at both the client and the relay were necessary. The first thing was to consider what information to collect and how. The main options we considered were to do logging at the network level or at the level of the Tor software. Since our thesis concerns an adversary running a middle node, the adversary should have at his disposal all the information an actual Tor relay would have. Due to the TLS encryption on the channels between ORs and the public key encryption used for the payloads of cells, we decided to use the Tor software itself for recording data.

4.1.1 Setting up logging

Tor by itself already provides very extensive logging capabilities. Different levels of logging are available, from the high-level error logs to the very detailed debugging logs. The problem with the debugging logs was, however, that they recorded a lot of sensitive information, while still not

enough circuit-level information for our specific needs. As a result, we ended up using a function normally used for logging errors and calling it at selected positions in the source code. Naturally this resulted in the need to modify the source code and recompile Tor.

In a normally functioning Tor relay, this specific function logs very few events, and so it suited us well since by using it we did not need to trim or otherwise process the log files before extracting circuit information from them. We used it to record a log event consisting of the current date and a string of our choice, which we used to determine the event in question. What we recorded consisted of the timings of events related to managing circuits and opening and closing streams. The specific data points recorded for each circuit are given in the Measurements section (Chapter 4.2). The required source code modifications totalled roughly 30 lines of code.

Once we decided what information to record, the next step was to place appropriate hooks into the Tor source code to achieve this. Based on background information about the Tor design and a review of the Tor source code, we concluded that the best places for this logging would be at the different functions used to process incoming command and relay cells. In short, we recorded the arrival of Create, Created and Destroy command cells as well as when our relay sent a Destroy cell itself. We also recorded the arrival of Relay cells, and the type of those. More specific details on the exact source code modifications are available in the Appendix.

4.1.2 Configuration

Once the relevant logging was in place, the next step was to set up the two hosts. This was achieved through a few additional source code modifications, setting appropriate configuration options in the configuration file, and the use of a firewall.

Since we were forced to make our OR public, we needed to consider the possibility of it carrying some traffic belonging to other users. We decided to rely on multiple 'layers' of protection. First of all, at the middle node, we greatly reduced the probability of our relay routing traffic belonging to another user by limiting incoming and outgoing connections to a select few entry and exit nodes. We would then construct our circuits using only these four medium-bandwidth nodes. The limiting was done using `iptables` (iptables, n.d.), which can be used to configure incoming and outgoing firewall rules.

As a second layer of defence, we limited the amount of data collected and ensured it was processed before transferring it to our local host machine or analysing it in any way. While a new OR is included in only a small number of circuits and our firewall ensured that those would only succeed if the user happened to pick two of the four whitelisted nodes as his entry and exit nodes, we still had the very small probability of recording information about another user. As a final safeguard, we made the decision to refrain from logging sensitive information, such as IP addresses

of connected entry or exit nodes. All exact dates were removed and the dataset was processed to extract only characteristics of individual circuits.

Apart from configuring the middle node, we also needed to ensure that the client would build circuits through it. A typical solution would be the use of the Tor Control Protocol (TorProject, 2014), which provides an interface for both getting runtime information as well as directly inputting commands. One of those commands can be used to construct a circuit with a manually specified path. However, we concluded early on that by building circuits manually we would not achieve our goal since we are specifically interested in the behaviour and consistencies resulting from the way Tor by itself builds circuits over time.

Fortunately, the path selection process for a Tor node can be influenced via setting appropriate values in the torrc configuration file. Two settings, EntryNodes and ExitNodes, can be used to give Tor one or more relays which it should always use when construct circuits. Additionally, setting the StrictNodes option forces Tor to use only the given nodes for the respective positions. With these settings we configured the client to always use the previously defined four nodes for the entry and exit positions.

Unfortunately there is no such option for the middle node positon, and thus additional modifications to the source code were required at the client. We located a helper functions used for path selection, which is responsible for choosing an appropriate middle node. This function was modified to always return a pointer to our middle node. That is, every three-hop circuit would always be built through our node.

4.1.3 Test Setup

The test setup consisted of a Tor client building circuits for HTTP and DNS traffic according to the normal operating behaviour and a middle node relaying this traffic. Both hosts recorded all relevant circuit-level information. After each measurement period, the information was processed at both hosts to acquire information pertaining to each individual circuit. Since our client built circuits according to the regular Tor specification, our measurements contained both internal[9] and exit circuits. From these circuits, the exit circuits originating from the client were selected and used to extract features for our classifier.

Both hosts in the test setup were running a recompiled version of Tor 0.2.4.22 configured via different torrc configuration files. The client was installed on a version 14.04 Ubuntu Linux Server running as a virtual machine using Oracle VirtualBox Manager. The relay was hosted at a high-bandwidth (1 Gbps network link) Tor-friendly cloud hosting company, using the same

[9] In principal, these were observed only for a short period after start up, since eventually Tor 'learned' that our client did not use Hidden Services and thus did not predictively build internal circuits.

Ubuntu Server operating system. We hide the identity of the node as well as the hosting company in order to mitigate against any possible influences our measurements or the release of these data may have.

4.2 Measurements

The measurements were conducted in three periods, between 15 to 18 hours each. The relay was already running when the Tor client was launched and remained operational for a period after the client had been closed. This was done in order to observe circuit behaviour at start up and shutdown in case there were any identifiable patterns[10] there. Two of the measurements were conducted using more active client behaviour and one mimicking a slightly more passive user.

4.2.1 Sample traffic

To generate sample traffic, the client was instructed to download web pages using Wget (Wget, n.d.) via Proxychains (ProxyChains, n.d.) at random intervals form popular addresses. Wget is a command-line tool for retrieving files using e.g. HTTP, whereas proxychains allows us to tunnel TCP-connections through a proxy server. In this case, we used it to direct all Wget commands using the local Tor SOCKS proxy. All DNS queries were also directed through the Tor network.

Downloads were simulated using a bash script that picked a random website from a list of 84 popular websites[11] and downloaded the main page and associated content of that website. For each download the script picked a new website. To approximate the delay between different page loads and to prevent any patterns resulting from the way we generate the traffic, we introduced a waiting period between each download. This period was a random number of seconds between 0 and 10 for the first and third measurement period and between 0 and 120 for the second.

The goal here was to approximate the behaviour of a person using Tor for browsing websites. More specifically we wanted to especially simulate an appropriate frequency of downloading content from different websites, i.e. opening different TCP streams. We reasoned that a user browsing regular websites would open a connection to a specific website and, because of all the associated content[12], would also actually open other connections in rapid succession.

4.2.2 Collected data

At the end of the measurement period, logs were processed locally at both hosts to devise information relating to each observed circuit. This was done using a python script, consisting of

[10] In the end, we came to the conclusion that the first and last circuits of each session actually made our results worse.
[11] We used a list of top-100 websites from (Alexa, 2014) and removed duplicate entries belonging to the same organisation.
[12] Remotely hosted advertisements, images, etc.

approximately 100 lines of code, which first constructed a list of all unique circuit identifiers found from the log file. For each unique identifier, the script then determined all cells related to that circuit and recorded the following information:

- Circuit Identifier (unique for both directions)
- Incoming or outgoing direction
- Creation and destruction times
- Times for first[13] and last relay cells
- Total and active time
- Total volume of cells

All times were recorded with a resolution of one second, since this was the resolution used by the Tor error logging function. We did not use total cell volumes for the classification task, but they were helpful in distinguishing between different circuits that had been created at the same time by the client. Although we recorded the times for first and last relay cells on each circuit, this was not to discover patterns in the traffic. Our target was to learn how the transition from one circuit to another, i.e. when the Tor client rotates circuits, would appear at the circuit level.

Once both hosts had extracted the circuit information, data was downloaded from the relay and processed further at the client. At this point we also followed on our goal of concentrating only on exit circuit and removed any internal client circuits from this dataset. Actually, recognizing all the exit circuits created by our Tor client turned out to be a simple task. For each created circuit, one could observe two new "circuits" created in opposite directions, i.e. one incoming and immediately another outgoing.

These pairs could be automatically picked from among the other circuits, which seemed to be mostly internal testing circuits or incomplete circuits blocked by our firewall, based on the opening directions and traffic volumes associated with them. Since both circuit directions were effectively part of the same client circuit, and had matching timings, we considered them one circuit in our measurements. We also note that the final cross-reference was done by hand, comparing the create times observed at the client and at the relay. Features for the classifier were extracted from this final list of circuits.

[13] More precisely the second relay cell, since the first relay cell contained an EXTEND or EXTENDED command used to extend the circuit and immediately followed the observed creation time, thus providing no extra information.

4.3 Classification

We can infer some patterns, and even probabilities with simple statistics from the gathered data. However, since we suspected there might be additional relationships between circuits of the same user, we decided to use modern machine learning techniques to help discover relationships between different events. Also, the automatic classification of circuit pairs plays a significant part in our ultimate goal of distinguishing which circuits belong to the same user. We want the classifier to tell us if two given circuits belong to the same user or not. More specifically, we need it to perform a binary classification task.

For this classification task, we decided to use a Random Forest classifier based on our expectations of the data we would acquire from our measurements. We expected the data to be quite noisy and highly non-linear, and Random Forests produce good results in both situations. They can efficiently identify clusters that might cause problems for other types of classifiers. Additionally, an important reason was the previously discussed goal of learning what properties were significant in matching two circuits. The way trees in a Random Forest are built makes it easy to extract that information which can later be used to determine precisely what design choices in Tor circuit construction actually result in consistencies between different circuits of the same user.

4.3.1 Extracting features

To train and test our classifier, we first needed to extract features from the collected circuit data. Here we use the terms *sample* to refer to one data point that contains multiple *features* and describes the relationship between a pair of circuits. **Positive** samples are those that contain features created by comparing two circuits known to belong to the same user, whereas **negative** samples are those resulting from comparisons of known non-matching circuits. Each data point contains five different features, which are comparisons between the times of two events of two different circuits. For example, we acquire a Create cell to Create cell feature by subtracting the create time of one circuit from the create time of another circuit.

There are two main reasons for using these distance-based features derived from different circuit events. Firstly, we want the features to be as generalizable as possible, so that the classifier does not need to be trained again for e.g. each point in time. Secondly, we are not interested in finding traffic patterns, and thus we do not wish to use features that directly depend on the type of traffic transmitted on a circuit.

Another python script (90 lines of code) was used to run comparisons between different circuits and extract the desired features. The script selected one circuit at a time and compared it to all other circuits, apart from itself, one at a time. The only exceptions were the situations where one of the circuits did not contain the desired features; these were primarily the very last circuits

of a measurement session, which did not necessarily have an end time. In any case we ended up with roughly $n * (n - 1)$samples for n circuits[14]. The difference between values for two different circuits was computed to acquire each of the following five features:

- Create time (of first circuit) to create time (of second circuit)
- First relay cell (…) to first relay cell (…)
- Destroy cell (…) to create cell (…)
- Last cell (…) to first cell (…)
- Last cell (…) to create cell (…)

For the first two features, we directly computed the difference between the two corresponding values of two different circuits in our data set. For the other three features, we made the comparison between two different properties of a circuit. As an example we computed the difference between the time when another circuit was created and when another one was destroyed. We did not differentiate between the order in which the two circuits were compared to each other, in order to avoid missing some relationships as well as to make later use more straightforward.[15]

Once we had acquired the samples and corresponding features from the circuits we knew belonged to the same user, we assigned them a positive label (of 1). In addition to positive samples, the classifier also needed negative samples to not only learn which circuits match but to learn which do not. Since we did not record any actual user data in our measurements, we needed to generate these samples ourselves. We created different circuits, separated by random intervals in time, and then extracted features from these in the same as described above for positive samples. The goal was to simulate completely random, unrelated circuits.

To be more precise, we used the circuits from our first measurement to build different templates for circuits based on the recorded information. Each template consisted of a create time, a destroy time, and times for first and last cells. By retaining this template, but randomizing the creation times, we effectively create authentic circuits that were spread randomly in time. We used a large number of these circuits to represent individual non-related circuits that a middle node would see in a real world environment, and that we labelled as negative (of 0). The immediate drawback from this is that our unrelated circuits might not be as unrelated as we assume them to be, since they are ultimately based on circuits that actually belonged to the same user. The possible

[14] A few samples less, since we did not run the comparisons for the very last circuits of each measurement set.

[15] We did not know the exact ordering of different circuit events beforehand, especially with predictive circuit construction, so we did not wish to fix the feature which would determine the order of two compared circuits. Furthermore, we did not see any significant downsides to training the classifier to recognize comparisons that did not have a fixed order.

issues with using these types of randomized circuits compared to using real world data is discussed in section 5.4 Limitations.

4.3.2 Random Forest classifier

For our classification task, we decided to use a Random Forest implementation from the open-source Scikit-learn machine learning tool (scikit-learn, n.d.). The main reasons were that it was readily-available using python and on the surface seemed to provide the functionality we needed from our classifier. The implementation allows tuning certain forest parameters and provides methods for training and accessing the Random Forest, as well as attributes detailing properties of the trained forest. The classifier's part of the python scripts were implemented using 96 lines of code, out of which only 24 lines implemented the classifier itself. A further 150 lines were used for computing the results and plotting figures.

From the source code we see that the implementation uses axis-aligned weak learners to split samples at each split node. To determine the best split at each node we can use either a Gini impurity or an information gain metric. Individual trees are trained following our earlier general description using subsets of samples for each tree and subsets of features at each node. The specification, however, details that in order to arrive at the combined prediction, the implementation does not let each tree vote[16] for a single class and take an average over those, as is the case in the original publication (Breiman, 2001), but instead averages the probabilistic prediction of each tree.

The tuneable parameters we used in our work were: number of trees (100), metric for splitting samples (information gain), number of features to consider at each node (sqrt(5)), and the maximum depth of a tree ("none"). Other additional parameters included the minimum sample size required for a split to happen or required to persist after a split. For these, we used the default values. Since Random Forests are highly parallelizable[17], we could also tune the classifier to run on multiple CPU cores. Once we had a trained Random Forest classifier, we used Scikit-learn Metrics to produce the results and figures in the upcoming Results section (Chapter 4.4).

4.3.3 Training the classifiers

Once we had extracted our features, it was time to train our classifier using them. The Scikit-learn –implementation we used does not allow a lot of configuration, but we can control the number and shape of trees using the parameters defined in the previous section. We experimented with different forest sizes and discovered that while at first the prediction accuracy improved rapidly

[16] Here, "to vote" means to assign a discrete label to that sample
[17] Each tree can be trained individually on a different processing unit.

when adding trees to the forest, no further improvement was attained after the forest size approached 100 trees. We thus settled for exactly 100 estimators, and left the max_features and max_depth parameters at their default values. We used three different subsets of samples, which resulted in three different classifiers. The subsets were selected to represent circuits separated by different amounts of time.

For each set, we constructed two arrays: one with all the samples and features, and one with the corresponding class labels[18]. These arrays were shuffled and then split into training and testing sets, with a 2:1 split. Shuffling was used to get roughly the same proportion of positive and negative samples into each set. We tested the classifier with a different set of samples than with which we train it on, since we naturally wanted to know how it performed on unseen data, as well as observe possible imperfections such as over fitting.

After training a Random Forest using the previously defined parameters, we used our test set to evaluate the performance of our classifier(s). For each sample, we extracted the predicted class label as well as a pure probabilistic prediction our classifier had for that sample. In our case, the higher the value the more confident our classifier is that this sample is a true match.

We could use these probabilistic prediction values and different thresholds between 1 and 0 to evaluate the true and false positive rates for our classifiers. For each threshold, we would classify samples with prediction values higher than the threshold, as positive or negative, depending on which class had this probability[19]. For example, with a threshold of 0.9, only samples with a mean probability of 0.9–1 (for both classes) were classified as positive or negative, respectively. The labels for each classified sample were then compared to the known labels we had for those samples, and as a result we acquired the true and false positives for that specific threshold. Varying this threshold gave us different ratios of true/false positives which can be used to evaluate the performance of our classifier in different settings.

4.4 Results

In the following, we present results obtained from analysing the collected circuit data and the results from our trained classifiers. Statistical analysis is used to learn properties from the raw circuit data as well as the features extracted by comparing different circuits to each other. Classifier results are provided in the form of figures illustrating decision areas for two-dimensional feature spaces as well as Receiver Operating Characteristic curves. These results are provided for three different scenarios, with the circuit-pair being separated by either a maximum of 15 minutes, one hour, or

[18] 0 or 1, with 1 signifying that this samples represents a match.
[19] This probabilistic prediction value is specific for each class, which in our case means that each sample has two values, one describing the confidence with which it was assigned either a non-matching or a matching label.

15 hours. For classifiers, we also provide equal error rates and selected true/false positive ratios that will be used later on to compute performance in a real-world situation.

4.4.1 Statistical results

The three measurement periods lasted 48 hours in total and resulted in 261 measured circuits[20] in total. The respective values were 15, 18 and 15 hours and 76, 101 and 84 circuits for each session. Table 4.1 below details the lifetime of each observed circuit. Total lifetime is the time from circuit creation to circuit destruction, whereas active lifetime is the time between the first and last relay cells. The minimum, maximum, and median values are given, as well as the standard deviation to describe the variation present in the values. Median values are used to give an idea of 'expected' behaviour, instead of e.g. simple averaging, in order to avoid a few stray samples having too large of an effect on the final value.

Table 4.1: Circuit lifetimes and active times for the three conducted measurements.

	CIRCUIT TOTAL LIFETIME (SEC.)				CIRCUIT ACTIVE TIME (SEC.)			
	Min.	Max.	Med.	Stdev.	Min.	Max.	Med.	Stdev.
MEAS.1	324	3641	639	700	277	878	601	71
MEAS.2	611	3911	656	457	459	754	575	43
MEAS.3	306	3027	647	573	23	758	601	94

Additionally, for each measurement, the percentage of circuit active times between +/- 60 seconds from the median value were 65/76, 89/101, and 74/84, which correspond to percentages of 86%, 88%, and 88%.

The samples created from all the previous measurements were combined into a single collection of positive samples. To get an idea of the most typical value for a feature, we took the most distinguishable concentration, determined the centre point for that, and took all samples from an area 300 seconds[21] in both directions. Table 4.2 below lists the number of samples within each selection and a few properties pertaining to each such feature.

[20] 522 pairs of incoming and outgoing circuits.
[21] That is half of the expected circuit lifetime

Table 4.2: Statistics for the first identified cluster in the extracted features.

Feature	Samples	Mode	Med.	Stdev.
Create to Create	242	601	613	68.7
First cell to First cell	244	600	610	47.6
Destroy to Create	249	-25	-20	53.9
Last cell to First cell	244	0	4	48.5
Last cell to Create cell	253	0	1	61.2

For each feature, the mode (most typical value), median, and standard deviation were computed. We can see that all sample sets had a strong concentration around either 0, -25 or 600 seconds with a large subset of samples residing close to the modes of each set with a decreasing number of samples when approaching the edges of the selection area. To set this result into context, in an ideal situation we would expect to see no more than 258 samples[22] with a Create to Create value close to 600 seconds. This would be the situation where all consecutive observed circuits are created approximately 10 minutes within each other. In this light, it seems that just by statistical analysis it is possible to estimate certain patterns in circuits belonging to the same user.

4.4.2 Classification results

To do better than simple statistical observations, we used Random Forest classifiers to learn consistencies in our sample set and distinguish between matching and non-matching circuit-pairs.

Based on our statistical analysis of circuits and features, we expected the performance of a classifier to be vary depending on how far away the two compared circuits were from each other in time. To observe this variation we trained three different classifiers using three different subsets of samples. We learned from our statistical analysis that circuits could be expected to be built approximately 10 minutes from each other and as a result these three subsets consisted of circuits separated by a maximum of either 900, 3900, or 54300 seconds. The times correspond to one, six, and 90 circuit lifetimes with a safety margin of half a lifetime in order to catch stray samples but to avoid including samples belonging to the next expected circuits.

Consecutive circuits

For consecutive circuits, i.e. those separated by one circuit lifetime, the sample set consisted of 508 positive and 508 negative samples, which were used to train and test performance of the classifier with a 2:1 split. We begin by illustrating observed patterns in a single feature: the Create to Create time.

[22] Each measurement set of n circuits contains 2*(n – 1) pairs of consecutive circuits, with half of them negative and half of them positive. With three sets of 76, 101, and 84 circuits we get 258 possible pairs.

Figure 4.1 below plots the samples and decision regions for the Create to Create cell feature against a random value. The samples include *both* training and test samples, with the known positive samples represented by dots and the known negative samples by triangles. The decision regions represent the patterns that the Random Forest classifier has identified based on the *training* set. The regions are acquired by plotting each individual decision boundary for all of the 100 weak learners used in the classification. The darker the region, the more weak learners have labelled that region as positive, and the more certain the classifier is of this classification. Conversely, the lighter regions represent negative regions.

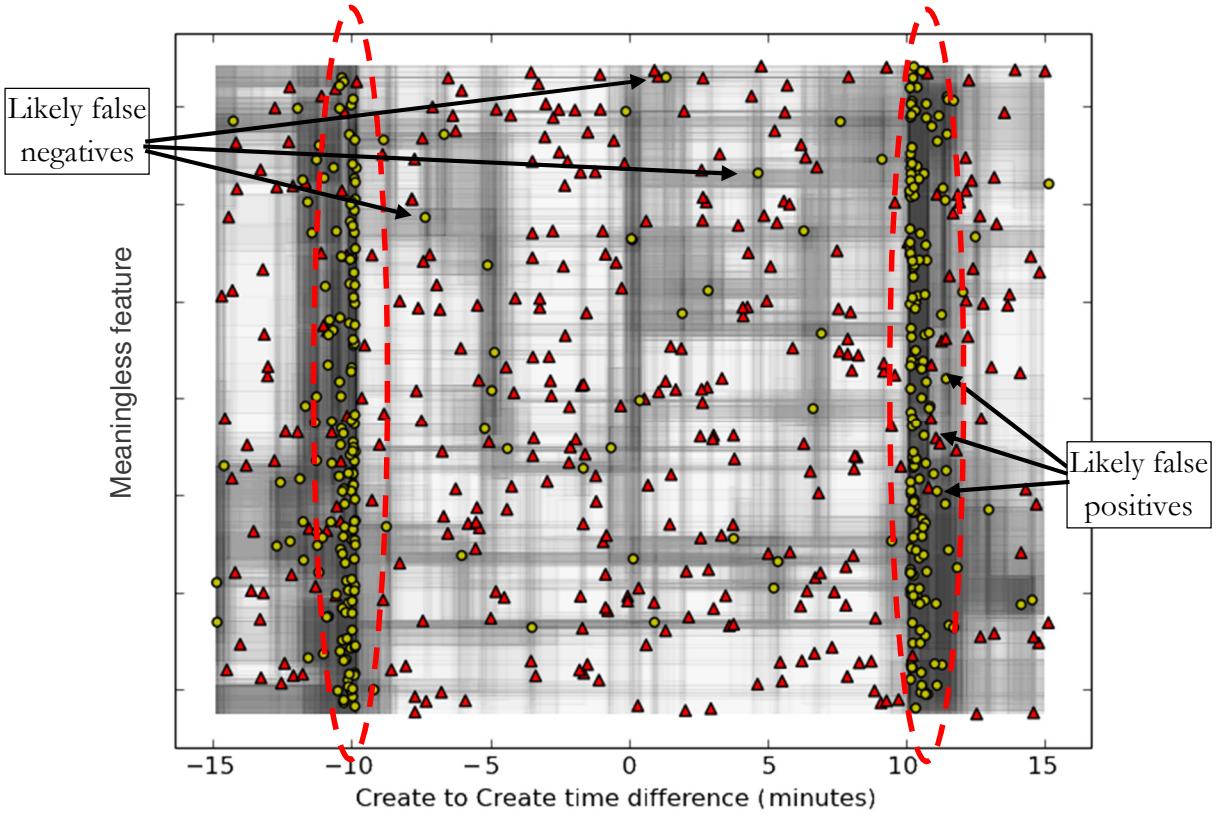

Figure 4.1: One meaningful feature. The known positive (dot) and negative (triangle) samples and the respective decision regions. Darker areas represent positive decision regions.

As expected, a clear majority of consecutive circuits are created within approximately ten minutes of each other. This can be seen as two clear vertical patterns of positive samples, which are highlighted. The resulting darker decision regions around the positive samples show that our classifier is able to successfully identify these patterns, and it also picks up some vertical patterns at -5 and 0 minutes. Conversely, we can see large light regions between -10 and 10 minutes, which the classifier has deemed to be mostly negative.

However, even if the classifier is able to classify the positive samples in the two large concentrations correctly, the large concentration of positive samples also means that any stray

negative samples (triangles) in those areas will very likely be classified as positive, resulting in false positives. The reverse is true for stray positive samples in the mostly white regions between or outside the two main patterns, which are likely to result in false negatives and a lower true positive rate. We have pointed out a few such samples for both situations.

The symmetry present in the positive samples stems in part from the way we extract features: two different samples are extracted from the same circuit pair since we do not differentiate between the temporal order of the circuits. In addition, the fact that the Scikit Random Forest implementation uses axis-aligned weak learners is apparent from the vertical or horizontal decision boundaries. Even though the second 'feature' is random, some of the weak learners use it for classifying some of the stray samples at the region between –10 and 10 minutes. This is not a problem since we use this random feature only for this single illustration and not for any of the classifications whose results we use later on. For reference, in this scenario the classifier achieves an Equal Error Rate (EER) of approximately 15.9%.

Moving on to two meaningful features, Figure 4.2 below again plots the known positive (dots) and negative (triangles) samples from both test and training sets together with acquired decision regions. This time we have, however, plotted two meaningful features against each other (Create to Create and Last cell to First cell). Now, using two meaningful features, the EER improves to 13.3%.

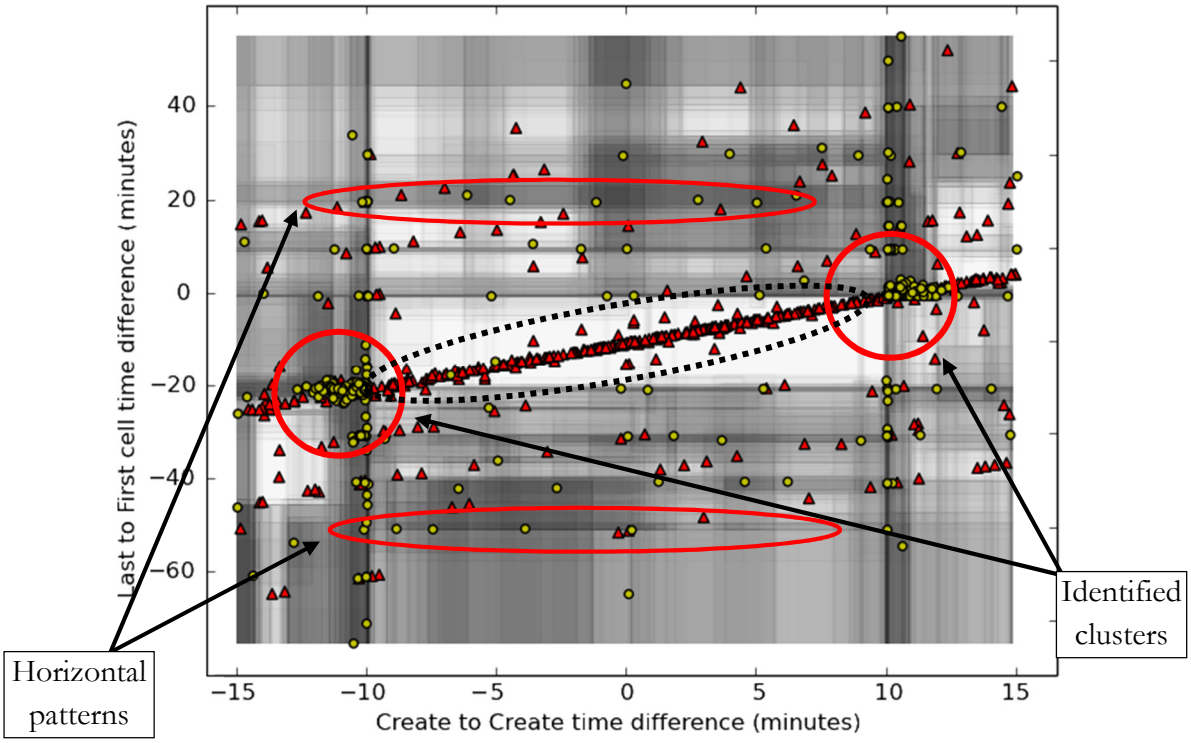

Figure 4.2: Two meaningful features. The known positive (dot) and negative (triangle) samples and the respective decision boundaries.

This time we can see two clearly identifiable clusters at around (–10 min , –20min) and (10min, 0min). Using both horizontal and vertical weak learners, the classifier is able to effectively identify these clusters, where most of the 508 positive samples are gathered. These two clusters are also highlighted in the Figure 4.2 above. For example the cluster at (10, 0) minutes implies that with two circuits built 10 minutes apart, traffic will typically start at the second circuit at the same time as traffic stops on the first one. We also observe some evidence of the predictive circuit building with the vertical patterns of positive samples at –10 and 10 minutes. They show that a circuit built 10 minutes after another one actually becomes active 10, 20 or other multiple-of-ten minutes later.

The classifier also very effectively identifies the pattern of negative (triangles) samples running between the two highlighted clusters. This is evident from the completely white negative region around it. What is interesting is that the classifier also picks out horizontal positive regions outside the two main clusters (and the vertical patterns): careful observation shows that the majority of positive samples, outside these areas, are actually not randomly scattered. There is clear horizontal grouping at multiples of 10 minutes, which contain the majority of these samples. As is evident from the slightly darker horizontal patterns, two of which are highlighted, the classifier is able to pick up some of these at least with some accuracy. More samples would probably make the classifier perform better here.

Finally, we look at performance with all five features. It is not possible to visualize so many features, so we use both the EER as well as a Receiver Operating Characteristic (ROC)–curve to evaluate the performance of the classifier.

The ROC curve for our classifier with all five features is plotted in Figure 4.3 below. The true/false positive ratio for different decision thresholds is plotted in blue, whereas the dashed line represents the performance we would achieve with simply guessing if the sample indicated a match or not. It works as a baseline that our classifier needs to beat at minimum. Ideally we would want to have as little false positives as possible and end up with an almost vertical line.

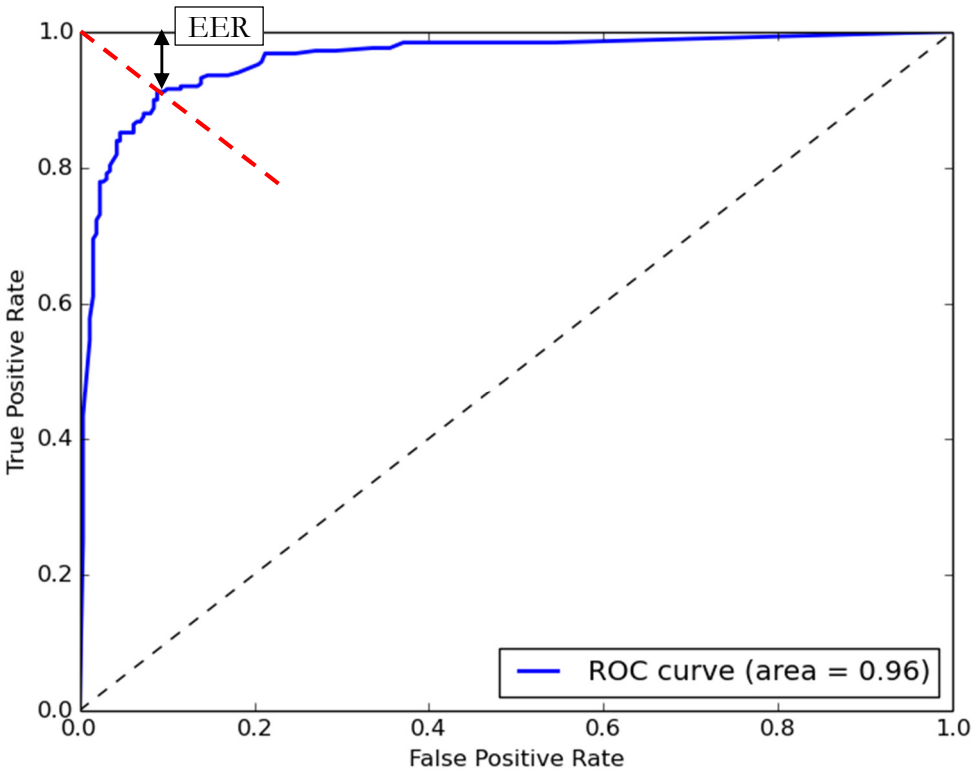

Figure 4.3: Receiver Operating Characteristic -curve for consecutive circuits using five features.

The threshold begins at one and results in very small true and false positive ratios, since with such a high threshold we are ignoring the majority of samples and selecting only those for which all the individual weak learners have been unanimous. When the threshold is lowered, we see that at first a larger number of positives are now classified correctly as positive and we achieve almost a true positive rate of 80% before our false positive rate starts to significantly increase. Intuitively, the more positives we can correctly identify without also getting a significant number of falsely matched negative samples at the same time, the better our classifier is.

The red line at the top left corner, plots the beginning of a false negative rate[23] vs. true positive rate curve, and it can be used to visually determine the EER. Since an EER is the situation when the false positive rate and the false negative rate are equal, we get the EER from the intersection of the ROC curve (FPR vs. TPR) and the line for (FNR vs. TPR).

Using all five features, our classifier clearly beats the performance of a random guess, even though the gradient falls quicker than one might hope at the top left corner. The achieved Area under the curve (AUC) is 96%, whereas the EER is 8.9%. While these metrics are good when comparing different classifiers to each other, in many practical situations they don't give us enough information about the characteristics of the classifier.

[23] False negative rate can be defined as (1 – True positive rate)

A very important performance indicator is, in fact, how the ROC curve behaves near the origin: a steeper curve indicates that we can achieve a higher true positive rate while maintaining a very small false positive rate. For a 50% true positive rate, our classifier requires a 0.8% false positive rate whereas a 5% false positive rate results in an 85% true positive rate. These will be important later on, when we wish to minimize the number of false positives.

4.4.3 Distant circuits

In addition to consecutive circuits, we were also interested in knowing how the results might change if the two circuits would be further apart in time. For this, the classifier was trained on 2784 samples consisting of circuits created within 3900 seconds within each other.

Figure 4.4 below represents a similar comparison to that of Figure 4.1 for samples with a maximum of one hour apart, in both directions. We plot only half of the available samples, in order to make the figure more readable. Create to Create time is again plotted against a random value, with dots representing previously known positive samples, triangles the negative samples and the greyscale areas in the background corresponding to the different decision regions.

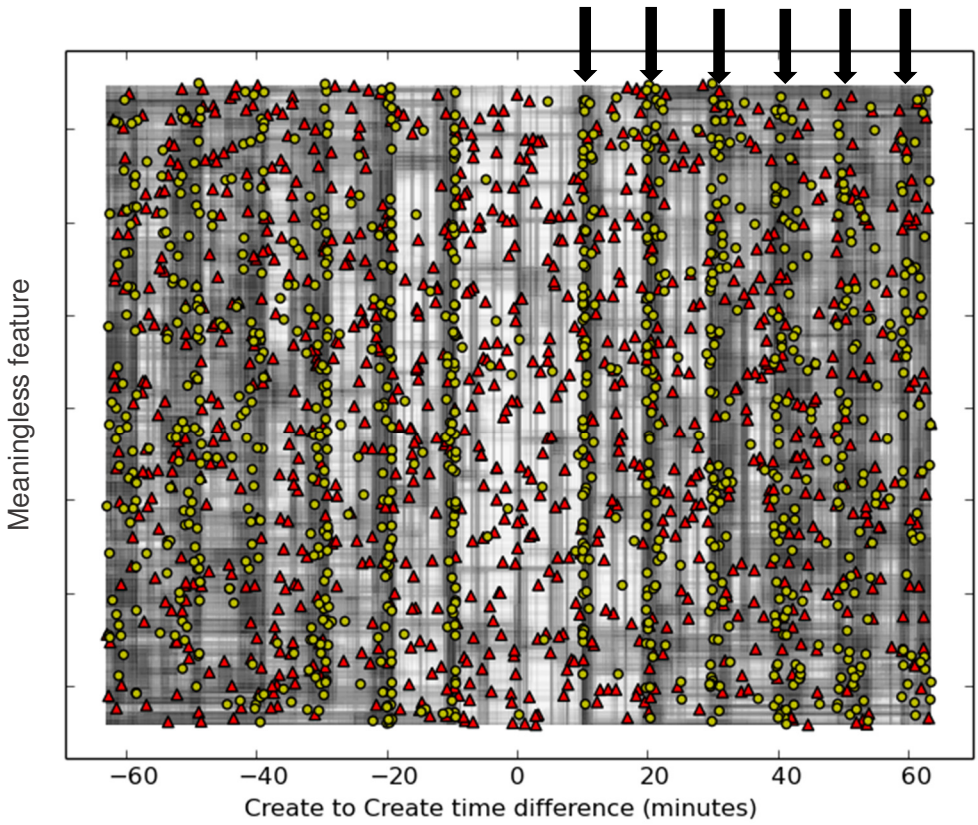

Figure 4.4: One meaningful feature for circuits created a max. of one hour apart from each other. Known positive and negative samples are represented by dots and the corresponding decision regions are plotted in the background.

Visible patterns are present at approximately 10 minutes apart from each other[24], as highlighted on the positive side by six arrows. We can however see that as we extend the distance between two circuits, the patterns become less clear. This is also reflected in the decision regions getting more spread out and uncertain. As a result an increasing number of samples are classified into wrong classes which leads to the performance deteriorating significantly from the situation of consecutive circuits. Here the classifier achieves only a 28.2% EER. The ROC curve for a situation with all five features in use is presented in Figure 4.5a.

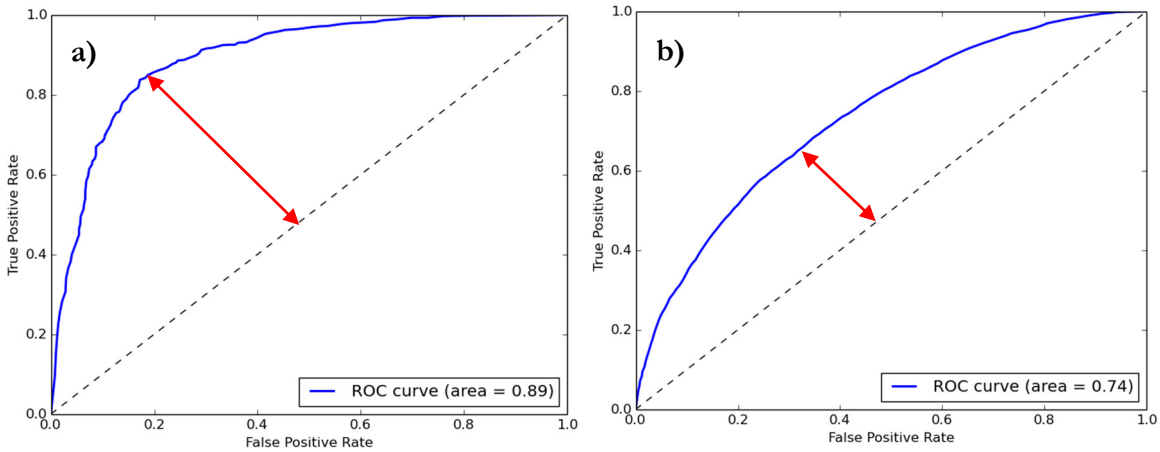

Figure 4.5: Receiver Operating Characteristic -curve for circuits using all five features and separated by a max. of a) one hour or b) 15 hours.

This ROC-curve confirms the performance reduction from the previous setting with an AUC of 89% and an EER of 17%. We see that the false positive rate increases more rapidly than for consecutive circuits and that the gradient at the beginning is much more gradual. Thus, a true positive rate of 50% is now achieved with a cost of 6% false positive rate, whilst a 1% false positive rate results in only a 15% true positive rate.

Figure 4.5b shows the performance reduction and the corresponding ROC curve when we increase the maximum separation to 15 hours. 22470 samples were acquired using a cut-off of 54300 seconds. The curve shows that with these distances the classifier no longer performs much better than if we were simply guessing the class of a sample, as is emphasized with the arrows. We achieve an AUC of 74% and an EER of 33.3%. At this point, a 19% false positive rate is required for achieving a 50% true positive rate, whereas a 1% false positive rate nets an 8% true positive rate.

[24] Except for 0 minutes, as is the case in all these figures, since we do not compare circuits to themselves.

5 Discussion

In this chapter, we analyse the results presented above and more importantly discuss their real-world implications. We then outline a possible attack strategy for an adversary as well as defences against our attack. Finally, we touch upon the limitations of our work. We analyse the numerical results, both statistical and classifier results, and explain what each result means. We then proceed to discussing and evaluating the threat posed to Tor users based on our findings.

We consider the probability of discovering a match in general without our attack, i.e. the base rate, and then combine this with our classifier results to arrive at a detection rate for an adversary wanting to link two circuits to the same user. We discuss the threat posed to users especially in a potential single guard node setting. As a separate issue, we determine what portion of the middle node traffic in Tor an adversary can feasibly control. We then combine the detection rate and controlled bandwidth share and present the main finding of our study: the probability for an adversary to link two consecutive circuits to the same user in a single guard setting.

Finally, we consider how an adversary could build an attack based on our findings, propose and evaluate how the Tor design could be modified to defend against this circuit linking, and consider the potential issues and places of improvement in our measurement data, classifier performance, and the limits on the generalization of our findings.

5.1 Result analysis

Our results show clear consistencies in the relationships between different circuits belonging to the same user. Also, the results for the Random Forest classifier seem very promising. In the following, we look at the numbers presented in the previous chapter, analyse their meaning, and draw conclusions from them.

5.1.1 Statistical results

Based on the recorded circuits, it seems that user exit circuits should be expected to remain open for around 10.5-11 minutes, slightly longer than one would assume simply by the rotation period. However, the active time remains fairly consistent at 10 minutes, hinting that there might be a strong correlation between the last relay cell on one circuit, and the first one on another one. The absolute values vary between around five and 65 minutes for the total lifetime and from 23 seconds to around 15 minutes for the active times. This means that circuits may remain open for a long time, but they are not actually used for much more than the intended 10 minutes. The results

obtained point to strong consistency in the way Tor users create and use circuits and support our original hypothesis that the way circuits are built and rotated would result in identifiable patterns.

Two important points to note here are that we only observed exit circuits, which partly explains the high minimum lifetime, and that we used traffic that resulted in frequent new connections. On the other hand, when we consider regular web browsing, the usage pattern does not seem to affect the circuit lifetime as much as one might assume. The second measurement was carried out using a maximum two minute waiting period between new downloads and despite this there is no noticeable increase in either lifetime. On the contrary, the median circuit active time actually decreased from the two other measurements.

The statistics gathered form positive samples confirm the expectation that there is indeed a strong consistency between the last cell of a given circuit and the first cell of the 'next' one for that user. Moreover, the values presented in Table 4.2 suggest that there would also be a strong relationship between the other chosen features. As expected, two consecutive circuits are created 10 minutes apart from each other and traffic flow begins at a similar interval. In addition, a new circuit is built and the same or another one is taken up almost immediately after traffic ends on one circuit.

What we find surprising in our results is the limited number of predictively built circuits. There are instances where a circuit is created well in advance, and where the time between the Create cell and first Relay cell is ten minutes or more. However, the majority of circuits contain the following start up pattern: the circuit is created and almost instantly we start observing a large and constant volume of traffic in the form of relay cells. We do not expect that this has any major impact on our results, but nevertheless we discuss the possible implications in the Limitations section (Chapter 5.4).

5.1.2 Classifier results

The results obtained with the trained Random Forest classifier confirm the intuition arising from the previous statistical analysis and further support our original hypothesis. As illustrated in Figures 4.1, 4.2 and 4.4 in the previous chapter, we can observe clear patterns that the classifier is also able to clearly identify. The achieved equal error rates can be used to roughly estimate the improvement achieved through using multiple features. The EER decreases from 15.9% to 13.3% when using two meaningful features instead of just the create to create time, and is further improved to 8.9% when the classifier can use all five available features.

In order to evaluate how good this EER value is, we need to first consider what task our classifier needs to perform. An EER is just one possible metric to evaluate our performance, and appropriate levels of true and false positives are often very dependent on the specific situation. In

the case of identifying circuits belonging to a single user from the Tor network, we can be fairly certain in expecting that our input will consist of a large majority of negative samples, i.e. non-matches. While we could be fairly certain (91.1%) that we would flag the few related circuits as matches, an 8.9% false positive rate would result in an overwhelming number of false positives.

For that reason, the more important result for the classifier is that it is able to achieve a 50% true positive rate with a low 0.8% false positive rate. This means that we will miss half of the matching circuits but if this avoids us following on huge numbers of false matches, it might be worth the trade-off. Although, the classifier would still flag 8 circuit pairs out of every 1000 false combinations as positive.

While the results confirm that there is a strong relationship between two consecutive circuits of a given user, they also show that this relationship diminishes when we need to compare circuits separated by more than 10 minutes. For circuits separated by 0-65 minutes our classifier now needs a 6% false positive rate for even a 50% true positive rate, whereas to achieve similar false positive rates as with consecutive circuits, the true positives fall as low as 15%. The situation deteriorates further when we try to match circuits separated by a maximum of 15 hours and we can no longer do much better than guessing. This all means that we would either miss most of our matching circuits or then we would be increasingly flooded with false positives.

What we can conclude is that while there are very strong consistencies between consecutive circuits, they are not so consistent that they would carry over to circuits far away. The small variations between e.g. the creation times of different circuits compound over time and result in too much noise for the classifier. Additional information would thus be needed in order to match these distant circuits to each other.

False positives and negatives

Our classifier produced some false positives and false negatives, and we wanted to know if there was a specific reason behind these incorrect classifications. We consider here shortly our findings, and how we could improve upon this.

We inspected incorrect classifications in two ways: extracting the specific samples that the classifier labelled incorrectly and plotting them with the decision boundaries we obtained for our Figures 4.2 (Chapter 4). Visual inspection helped us locate clear areas where the classification results were off and we could then inspect the specific features of these samples. The identified samples confirmed our expectations from Chapter 4.4 that the large concentrations of positive samples would result in some negative samples in those areas being classified incorrectly, and that stray positive samples outside the main clusters would not be correctly identified.

Using a decision threshold of 0.5, from the 17 observed false positive circuit pairs, 10 had absolute creation time differences close to 10 minutes and traffic end and begin times located very

close in time. In addition to these samples, 5 false positives had creation time differences of approximately 10 minutes and a further 2 pairs showed close matching in their traffic start and stop times. Thus all the incorrect classifications were because the samples had features similar to the vast majority of positive samples. For the false negatives the situation was slightly different: 9 out of 15 samples had traffic end-begin time differences of multiples of 10 minutes and four (of those nine) circuit pairs had been created at the same time.

We deduced the original circuits from these misclassified samples containing the described features to inspect what types of circuits these (false negative) samples were created from. The most prominent situation was that of when two circuits had been created at the same time, but one remained unused for a period of one, two or more circuit lifetimes. That is, when circuits are built predictively, and especially when they remain unused for a longer time than 10 minutes.

We can conclude from our findings that the majority of false positives results come from non-matching pairs that happen to exhibit the same features that truly matching pairs do. One explanation could be that our generated random circuits are not as unrelated as we perhaps hoped. However, if we were to observe enough (non-related) circuits in the live Tor network, a fraction of those circuits would inevitably have properties that would make them impossible to distinguish from the case where the circuits truly matched.

When it comes to false negatives, it seems that the results were a result on the classifier "over-relying" on the creation time feature and thus overlooking samples that had a distinguishable traffic end-begin time feature. One reason for this was the relatively small number of samples we collected and used for training the classifier. Given more samples, it might better pick up these patterns related to the traffic end and begin times. This was, in fact, something we noted when discussing the properties of Figure 4.2 (Chapter 4).

5.2 Implications of results

We have determined that the different circuits of a Tor user can be effectively linked together, with a certain ratio of true and false positive rates. However, we have yet to show what, if any, threat this poses to a Tor user. Two things need to be considered: what is the detection accuracy for an adversary and how many circuits is he able to observe in the network? The former depends greatly on the set of potential circuits we need to consider. To this end, the number of entry guards plays a significant part.

We will thus present the main result of our work: the rate at which the classifier will indicate a positive match to the adversary for an observed circuit pair. We begin our derivation by considering a global adversary who can observe traffic at all middle nodes. For different situations

we have different *base rates*, i.e. expected number of observed positive matches. Finally, we will estimate the probability for an adversary controlling a subset of middle node traffic.

5.2.1 Base rate fallacy

A significant issue when considering how our results translate into real-world performance is a so called *Base rate fallacy*. The base rate determines the occurrence of a certain event, such as one in every hundred, and base rate fallacy concerns the situation where this rate is very unbalanced. That means that we receive very biased information. This is relevant in our situation, where we wish to find a matching for a given circuit out of a large set of unknown circuits, i.e. our information is very biased towards non-matches. Even if our classifier is very good and has a low false positive rate, if we run our comparison on a huge number of non-matching circuits, we will still end up with so many false positives that they overwhelm the small minority of true positives we accurately match.

We use Bayesian statistics, and more specifically the *Bayesian detection rate* to come up with the rate at which our classifier will produce a positive match for an observed circuit pair. The detection rate is derived from Bayes' Theorem, which gives the relationship between conditional probabilities. It is widely used in statistics and machine learning to discover the posterior probability given some prior probability and likelihood. We will use it to transform our results from our specific setting into the general case.

To begin, we determine the symbols used and relate them to the properties of our classifier:

- M = "The event of two matching circuits"
- ¬M = "The event of two non-matching circuits"
- D = "The event of the classifier predicting a match"
- ¬D = "The event of the classifier predicting a non-match"

P(D | M) is the probability of D being true given that M is true, and in this case represents the **True positive rate**, i.e. the event of our classifier predicting a match given that the two circuits match. In a similar fashion we get the **False negative rate** P(¬D | M) = 1 – P(D | M), the **False positive rate** (D | ¬M) and the **True negative rate** P(¬D | ¬M) = 1 – P(D | ¬M).

Bayes' theorem, in its simplest form is:

$$P(M|D) = \frac{P(D|M) * P(M)}{P(D)}$$

Since $P(D) = P(D \cap M) + P(D \cap \neg M) = P(M) * P(D|M) + P(\neg \mathrm{M}) * (P(D|\neg \mathrm{M})$, we get:

$$P(M|D) = \frac{P(M) * P(D|M)}{P(M) * P(D|M) + (1 - P(\mathrm{M})) * (P(D|\neg \mathrm{M})}$$

which is also known as the Bayesian detection rate.

Before we can determine the detection rate using our true and false positive rates, we need to determine our base rate, i.e. the probability $P(M)$. This is calculated as the number of matching circuits divided by the number of total circuits from which to choose. For this we need to determine the total number of circuits we expect to encounter.

Let us assume a global adversary who observers all middle nodes and sees all traffic coming from entry nodes. Using Tor metrics (TorMetrics, n.d.), there are approximately two million connected Tor users at any time. Let us estimate that one fourth of those would be actively using Tor, which would result in 500'000 users building circuits at any given time. Assuming we do our match on a user who builds the same number of circuits as an average user in the Tor network, we can say that we have a prior probability of 1/500'000 in this situation

However, the adversary can also derive significant additional information from the knowledge of which entry node a given circuit comes from. Each user has his set of three guard nodes, which remain constant for a relatively long period of time. If the adversary knows these three nodes, the base rate is dependent only on the number of possible users using these three entry guards. Even if the middle node has no way of determining all three guard nodes, it will still be able to observe on average one third of the circuits created by the user.

Using the current directory consensus and available relay descriptors we can, using a modified version of the code by (Kadianakis, n.d.) for computing guard node cut-off bandwidths, estimate the expected number of users per one guard node. Using the current 2 MB/s[25] guard bandwidth limit and assuming the same 500'000 users constructing circuits, we get an estimation of 190 users per the median guard node in the network. This figure gives us a rough estimate of the number of possible users we need to consider for each connected guard node.

In the three guard node situation, we have narrowed down the set of possible circuits from 500'000 to 570. Running the calculations for this prior probability gives us:

$$P(M|D) = \frac{\frac{1}{570} * 0.5}{\frac{1}{570} * 0.5 + \left(1 - \frac{1}{570}\right) * 0.008} \approx 9.90\%$$

A reasonable detection rate, considering that we will have a true match for every ten circuits our classifier flags as positive. If we compute the same detection rate using the base rate of 1/500'000, we achieve a detection rate of only 0.0125%. This clearly illustrates the significance of always taking the base rate into account.

[25] Increased recently from 125KB/s to 2MB/s for the tor-0.2.5.6-alpha version.

5.2.2 Single guard node

The plans for moving to a single node system make this situation much more dangerous. As shortly mentioned in Chapter 3, in order to protect against fingerprinting and the threat of eventually choosing a malicious node as the entry or exit node, Dingledine et al. (Dingledine & al., 2014) propose that Tor users would use a single preferential guard node for nine months at a time. They show that, while an unlucky node that picks a malicious entry node is hopelessly exposed, overall the probability of a user picking a malicious entry node is reduced, especially over time. However, we will now consider what this change would mean regarding our attack.

Using only a single guard results in a drop in the base rate which directly affects the achievable detection rate. In addition to the case of a median guard node with 190 users, we also now consider a worst-case scenario where the user picks a small node as his guard, i.e. one of the 2MB/s guard nodes. The expected number of clients per such a node is just 13.

Using these values, the detection rates become 24.85% and 83.89% for the median and the worst case. For the median situation, we need to process approximately four circuits to find a true match, whereas for the worst case situation, more than eight out of ten pairs flagged as matching will truly belong to the same user.

These are both very strong results and especially troubling for users picking a slow guard node. Since our base rate is now relatively low, we can also consider the situation where we tune our classifier for a higher true positive rate in order to find a larger proportion of the matching circuits. In the results section, we saw that the EER for our classifier is 8.9%, which implies an 8.9% false positive rate and a 91.1% true positive rate. When we use these values for the case of a small single guard, we achieve a detection rate of 46.03%. This means that while we now get two positive results per each truly matching circuit pair we now also catch 91.1% of the matching pairs instead of 50%.

While the previous values are all calculated for a situation where we are matching two consecutive circuits, the situation with circuits further apart from each other can also be relevant. With a maximum one-hour separation our classifier achieved a 50% true positive rate with a 6% false positives. For the two different entry guard sizes, this works out to detection rates of 1.44% (69 circuits) and 40.98% (2.4 circuits). This result shows that matching distant circuits becomes significantly costlier for the average case. However, the worst case result still remains strong as we would still need to follow up on only between two and three circuits to find a truly matching pair. This does not mean that we can match a circuit pair one hour apart with these detection rates, since the true/false positive rates above are calculated for the *whole* range of 0…60 minutes, and so performance at the very edge of this reason would worse than the rate used here.

5.2.3 Observable bandwidth

The final piece in evaluating our attack is to determine the relationship between the prediction accuracy of the adversary and the portion of circuits he is able to observe. We are interested in the probability with which a client picks a middle node controlled by the adversary. To this end, we take available bandwidth information from Tor metrics (TorMetrics, n.d.) and combine it with knowledge of the weights Tor assigns to each type of node when selecting nodes for a circuit path.

While client decisions are based on the advertised (measured) bandwidth values of each node, unfortunately Tor metrics does not provide this data according to different types of relays. As a result our calculations are based on the available bandwidth data based on actually transmitted traffic. Despite this we hope to give a rough estimate of how much circuits an adversary may hope to observe, and at what cost.

Tor uses different weights for each type of relay for each type of position, varying them depending on the total network status, with the goal of steering nodes to be used in those positions where there is the most need for them. Currently, based on Tor metrics, the total measured network bandwidth is approximately 5050 MiB/s, and it is divided between relays with different flags in the following way: exit bandwidth 250 MiB/s, middle[26] bandwidth 450 MiB/s, guard bandwidth 2700 MiB/s, and finally guard and exit bandwidth 1650 MiB/s. The last one is the set of routers containing both flags.

This results in a shortage[27] of exit bandwidth, which results in a weighting that favours the selection of nodes with exit or both exit and guard flags for exit node positions. The flipside of this is that these nodes are weighted less for the middle position. Computing the weights according to (TorProjet, 2014) results in a 0.419 weight for a guard to be chosen as a middle node, a 0.056 weight for choosing a guard and exit flagged relay, and weights of zero and one for exit and un-flagged (middle) relays respectively. As a result, e.g. a middle node will be chosen slightly more than twice as often for the middle node position than a similarly sized guard node, whereas an exit node will never be selected as a middle node, according to this weighting.

We can use this information to estimate the fraction of the circuits an adversary will observe if he controls a certain percentage of the middle node capacity. Because the middle node has the best weight for the position, we assume the adversary controls a set of middle nodes.

[26] Bandwidth for those relays that have neither an Exit nor a Guard flag.
[27] Guard or Exit bandwidth is *scarce* if either account for less than 1/3 of the total network bandwidth.

Setting x as the fraction of total middle node bandwidth in MiB/s, we get the following equation:

$$\frac{M * x}{E + G + E\&G + M + M * x)} = \frac{450 * x}{1675 + 450 * x}$$

where M, E, G, and E&G correspond to the Middle, Exit, Guard, and Exit+Guard bandwidth classes. We see that the node weighting ensures that in order to observe a certain fraction of the middle node traffic one does not need to control an equivalent share of the total network bandwidth. E.g. observing one third of middle node traffic is possible by controlling less than a third of the overall network bandwidth. As an example, adding the equivalent of 1% of the current middle node bandwidth (4.5 MiB/s) into the network would result in observing 0.27% of the circuits. Furthermore, adding 5%, 10%, 50%, 100%, and 370% would net control over 1.3%, 2.6%, 11.8%, 21.2%, and 50% of middle node traffic respectively.

Assuming a cost of £0.50 per Mbps (per month) (VPS9, 2014), observing about one fifth of the circuits at any one time would cost the adversary roughly £1'900 in bandwidth per month. To observe half of the constructed circuits would still incur 'only' a cost of £7'000 per month. This should be considered only as a rough estimate, since the cost per Mbps is based on advertised pricings and bandwidths, so it is possible the actual costs would end up being higher.

However, we can still deduce that it is not out of the reach of an adversary to observe half of the middle node traffic. We can thus compute a probability of an adversary linking two circuits belonging to the same user. Looking at the case of an unlucky user picking a small node in a single guard setup, our adversary will match 91% of the truly matching circuits, although he will need to consider roughly double that number of circuits since his detection rate is 46%.

Taking everything together, for a given circuit, on average our adversary will find the next circuit the same user uses with a probability of roughly 45% but needs to follow on average two circuits to discover which one truly belongs to the user.

5.2.4 Extending our attack

We can also vision a possible attack strategy that an adversary could take for extending beyond the attack that we have detailed thus far. In the previous section, we gave detection rates and probabilities for situations where the adversary wants to match a single pair of circuits to the same user. In reality, however, the adversary would most likely want to seek to link multiple circuits to observe the target user for perhaps the whole length of his communication session. We present here an outline for an adversary who wants seeks to link long chains of circuits and use comparisons between multiple circuits to achieve higher accuracy for each individual circuit.

Instead of looking only at the match between one circuit pair at a time, the adversary would build complete chains of circuits by looking for matches among a previously unknown circuit and

all previously identified circuits. He would begin with one circuit and discover all matching circuits within n circuit lifetimes of this circuit. Since the accuracy decreases with circuit separation, he will find the more matches the further away he goes. However, for the circuits close by, he will have less possible matches.

The adversary would next select the nearest match in time and proceed to discovering all matches for this node. Now, since there is a significant correlation between circuits even a few circuit lifetimes away, some of these matches will necessarily be with the same circuits that the original circuit also discovered. If our classifier tells us that one circuit belongs to the same user as two other circuits, which have already been flagged as matching, our confidence will increase that this third circuit would also belong to the same user. Repeating the previous process, the adversary will discover more and more matches for one individual circuit. The closer the two matching circuits are in time, the higher their importance would be.

Depending on the desired accuracy, the adversary can use the acquired information to construct complete chains of circuits or individual fragments from the circuits he has found the most matches for. As a result he is able to profile the user and analyse his traffic over a long period of time. If at some point the user has leaked some identity information or e.g. accessed some sensitive service or data, all his other traffic can be associated with this information.

5.3 Suggested Defences

We have shown that it is possible for an adversary to link different circuits to the same user because of the consistencies resulting from the way Tor currently creates and uses circuits. Also, the current weighting of node selection and the absence of safeguards against middle node misuse makes it more feasible to control the middle position of a circuit. Since our ultimate objective is to improve the security of Tor, we will briefly consider a few possible defences against our attack and evaluate their effect.

We see two main ways of reducing the accuracy of an adversary matching two circuits. The obvious modification would seem to be randomizing the circuit rotation period to remove the distinct relationship between circuit creation times. Unfortunately, it is unlikely to have a significant effect on the accuracy of the prediction, and in the worst case, if done wrong, can actually make it easier to match circuits. The wrong way of doing this would be to randomize the rotation period, but only at Tor start up. This would result in each user (session) having a distinct identifier that would significantly reduce the base rate if an adversary were to discover this quasi-identifier.

Even if the rotation period is randomized for each circuit individually, it does not necessarily mean that the adversary could no longer accurately match. The reason for this comes from

observing what features our classifier deemed as most influential in the classification. Even though there was a strong correlation between circuit creation times (at intervals of 10 minutes), there was an even stronger correlation between the time one circuit became inactive and another one active. Randomizing the creation times of circuits would break this relationship between circuits separated by more than one circuit lifetime, but would not affect the strong relationship between two consecutive circuits.

Hence, even though randomized circuit lifetimes would improve the situation for distant circuits, it would not defend against linking two consecutive circuits. It is a good starting point, but more is needed to break the relationship between the uses of consecutive circuits.

When it comes to other types of defences, one possible mitigation is to make it harder for an adversary to control the middle node position. The issue here is that these nodes are already not eligible for any other position in the circuit, and if they are not accepted even as middle nodes, the Tor network loses their contribution completely. Currently, there seems to be an abundance of guard capacity (TorMetrics, n.d.) so one way would be to tweak the weights so that guards are more often considered for the middle position. However, this would not significantly affect resources required to control a certain fraction of middle node traffic.

One major way of reducing attack efficiency would be to influence the base rate. Our attack is most effective in a single guard situation, where the expected occurrence rate of positive matches is significantly higher than in the current three node configuration. Perhaps the results presented in this work will prompt a reconsideration of implementing this change. The benefits of moving to a single guard node would be dwarfed by the threat of an adversary observing complete user sessions via linking his circuits together.

5.4 Limitations

It is important to also discuss the limitations of our work. We consider here three types of limitations: those affecting the legitimacy of our collected data, the performance of our classifier as well as issues we have not considered in our work.

Collected data

First of all, there are certain limitations regarding the data we have collected. These relate to the used setup and the way negative samples are created. Since we wanted to avoid collecting any sensitive information and were not able to run our middle node as a private node, the setup we used did not fully represent a real situation in the Tor network. This may have made our data less noisy than when circuits co-exist with other traffic in the network. Another factor here is that we

only recorded traffic from one client, and collecting data from multiple different ones could also have added noise to the data.

In addition, the way we limited our circuits to a few entry and exit nodes seems to have affected the predictive nature of circuit creation. While some circuits were clearly built proactively, the majority are not. In a separate (independent) test, with relaxed limitations on entry and exit nodes, we could observe a slight increase in the number of predictively built circuits, although this increase was not significant. We could not find an explanation for this behaviour from the specifications or source code, and it might be that this behaviour is consistent with what one would observe in a normal situation.

Another result of our decision to not record data on actual Tor traffic and circuits was that we did not acquire negative samples from real data. Instead, we built templates out of our own recorded circuits, added random offsets and used those new circuits as random circuits. Certain properties, however, still persisted and as a result our negative samples might not have actually been as unrelated as real circuits would have been. For example different delays associated with opening streams for a new circuit, or even the circuit lifetime, would probably not remain as constant as they were in our data. This is something that might have had an effect on the accuracy of our classifier since in that case it would not have picked up on some patterns that should have been unique only to the positive samples.

While the aforementioned issues could have made our data more ideal than in the real Tor network, for both positive and negative samples, we do not see that they would significantly alter our results. Since the most significant feature turned out to be difference between traffic ending on one circuit and beginning on another, this limits the effect a more noisy create time would have on the classification. Moreover, having more diverse non-matching circuits could in fact steer the accuracy in the other direction.

Classification

We have three limitations as to our classification accuracy. First of all, Random Forest classifiers perform very well when presented with a very large number of features. Since we used only five features, we did not take full advantage of the capabilities of our classifiers. Extracting a larger number of features from the dataset could have resulted in better prediction accuracy. A related issue is the number of samples we used for training our classifier. We noted in our analysis of the causes of false negatives that our classifier could not pick up certain patterns most likely due to the small amount of training samples in those regions. More samples could have resulted in improved true positive rate.

Finally, the Scikit-learn implementation of Random Forests could have resulted in slightly less than optimal performance. More specifically, the axis-aligned weak learners were not the ideal

choice for our clustered data sets. We could have possibly achieved better results through the use of another type of weak-learner such as a general oriented, a quadratic, or even a non-linear one, which could have isolated the clusters more accurately.

General applicability

We also acknowledge that we have deliberately limited the scope and thus the generalization and applicability of our work. This has to do with the type of circuits we observed, the data we collected and the features we extracted as well as the adversary that we have considered.

We made a conscious decision to observe only exit circuits and ignore any internal circuits to hidden services. It can be expected that these circuits would contain similar consistencies to exit circuits but it is not clear if linking them would be easier or harder. Another limitation is that we didn't use any long-lived ports in our measurements. However, in the case of these long sessions using a single circuit, an adversary on the path could already observe the traffic for an extended period of time, and there would be no need to link any circuits. On the other hand, even if the user would rotate to new circuits at some point in time, our classifier might be able to pick that up based on the end and begin times of traffic on these two circuits.

Another deliberate limitation on our results is that we decided to look at only circuit events and begin and end times of traffic. Doing this we set a lower bound for the detection rate of an adversary, but also limited our ability to evaluate the actual threat to a Tor user. To do this, one would need to consider the information gain achieved with our attack and combine that with any additional information leaked from user traffic.

Finally, we have only considered an adversary who wishes to link circuits to the user one pair at a time. This is certainly not the only or even the most effective way of linking circuits, as we shortly discussed in section 5.2.4. We have also not considered what other information the adversary might have or what other parts of the network he could control. All this is not part of our work because our goal was not to ultimately break the anonymity of any real user but instead to expose a previously unknown threat so that it can be mitigated before anyone actually devises an attack around it.

6 Conclusions

We have presented a novel attack against Tor by showing that an adversary can successfully link different circuits to the same user, while using only timing information on circuit construction and use. We have shown that in a single guard setting a moderately resourced adversary can very efficiently identify circuit pairs belonging to the same user and use this to learn whole communication sessions of that user. Although on its own this information does not allow an adversary to discover the actual identity of the user, it can be used to greatly degrade the provided anonymity. Using our attack, an adversary can observe a users' traffic and profile him for much longer than the intended 10 minutes thought to result from rotating to new circuits.

Through analysing the Tor specification and source code, we set up an environment and recorded data for the circuit construction of a user. We also successfully identified features that describe the consistencies between different circuits belonging to the same user. We use a state-of-the-art machine learning technique, Random Forest classification, to determine how accurately an adversary can distinguish between matching and non-matching circuits.

Our results and the subsequent analysis show that an adversary can gain a significant advantage from observing the timing of events related to circuit construction and use. This advantage on its own is not enough to match circuits at the level of the whole network, but if we take into account the use of a limited number of guard nodes, we see that the detection rates become more feasible for an adversary. Even in the current three guard node setup, our attack combined with other information, or even on its own, poses a threat to Tor users.

Especially troubling is the finding for a single guard node situation. In light of our attack, it seems that it would be good to reconsider the risks and benefits of moving to a single guard system. We consider some possible defences against our attack, but it seems that in the short-term this is the only way to effectively mitigate the threat. We note that simply randomizing the lifetime of circuits would not necessarily solve the issue, since the most important feature in matching circuits was the time difference between the traffic ending on one circuit and beginning on another one.

In conclusion, we have presented a novel attack which, to our knowledge, is the first attack that successfully links together different circuits belonging to the same user, using only timings of circuit-level events. Due to the discovered threat, it is necessary to devise new defences and reconsider the plans regarding single guards. We have also brought new insight into the actual threat posed by the middle node in a Tor circuit, which will hopefully result in more attention to its role in the future.

6.1 Future work

Regarding possible future research around this topic, we see a few possible directions. These are related to improving the accuracy of our classification, analysing the effects of possible defences and also, devising further attacks based on our discoveries.

First of all, our measurements and classification had two main limitations: our negative samples were not based on actual user circuits and we did not take full advantage of the capabilities of Random Forest classification. We did not want to risk recording any sensitive information, but if one were to use sufficient sanitisation and with the permission of the Tor Project record data from the live network, it would be interesting to see how it would affect the accuracy of the classification. On the one hand, a possible increase in predictive circuit building could degrade the results or then acquiring more appropriate negative samples might actually improve them. A related issue would be to observe the performance of a Random Forest classifier on circuit-level data using more complex weak learners.

When it comes to defences, one direction for future work would be to evaluate the exact effect of randomizing circuit rotation times. We expect that the results would degrade, but not significantly enough to sufficiently mitigate the threat. It is thus necessary to in the meantime investigate other possible defences. Finally, we also see it possible to develop considerably more powerful attacks based on our findings. One such attack would be the one described in our analysis section, where the adversary creates chains of circuits by repeatedly matching individual circuits to multiply other already matched circuits. These attacks could significantly improve the advantage of an adversary and will need to be considered when designing any defences.

Acknowledgements

I would like to thank Dr George Danezis for the time and effort he has put into supervising my thesis. I am very grateful for all the ideas, comments and valuable insight he has shared with me. I also wish to thank my parents for supporting my studies and setting an example to follow.

Appendix: Source code modifications

In order to log necessary information and force the client to build circuits using a desired path we modified the Tor source code for a few different files, and set appropriate parameters in the configuration file(s).

Tor is written in the C programming language, and the source code and the various functions are divided into different c-files depending on what purpose each of the functions serves. Functions (and files) are named appropriately to make determining their purpose easy and to make determining appropriate points in the code straightforward. For our work, we modified four files in particular: channel, circuitbuild, command and relay .c-files. We also modified a few other files when researching why our middle node would not relay traffic as a private node, which are not relevant here.

For logging we used the built-in function `log_err()`. By default this function is used for error logging to a predefined file on disk. For each event it logs the current time, the type of log message (error, in this case), the function which is used to call it and a log message. We used this command message to a string we could later use to identify exactly what event had taken place.
Tor uses command cells to construct and tear-down individual circuits, so the natural places for the first hooks were the functions responsible for processing incoming control cells. Thanks to the clear naming used in the source code they were fairly easy to locate. Create cells are processed by the `command_process_create_cell()` function, created cells are handled by the `command_process_created_cell()` function, and so on.

In order to get the arrival time for each cell, a specific identifier of the form [PATTERN]:[COMMAND] and a circuit identifier from the structure of the incoming cell (`cell->circ_id`) was passed on to the error logging function at the beginning of each of these functions. PATTERN was used to identify the log events relevant to our measurements whereas the circuit identifier was used to identify all cells belonging to the same circuit.

In addition to command cells, it was necessary to record the presence of those relay cells used as part of circuit construction. To this effect similar identifiers were included into the `connection_edge_process_relay_cell()` function which is responsible for the processing of recognized[28] relay cells. The occurrence of each type of relay cell resulted in an appropriate event into the error log.

[28] Relay cells which have be successfully decrypted and are thus destined for the relay in question.

These two sets of modifications covered all desired events for both incoming and outgoing circuit-hops for both client and relay, except in one situation: when the relay wants to destroy a circuit. This happens when a circuit is torn down, one hop at a time; the OR does receive a DESTROY cell on the circuit facing the client direction, but it is itself responsible for sending a DESTROY cell to the other direction. This event is handled by the `channel_send_destroy()` function and a logging function was thus added there as well.

The final source code modification was to force the client to build circuits through our middle node. The other nodes we could configure from via the `torrc` configuration file, but unfortunately there is no such option for the middle node positon. We identified a helper function `choose_good_middle_server()`, and modified it to always return a pointer to our middle node.

The following functions were modified to record information:

Channel.c

int channel_send_destroy(circid_t circ_id, channel_t *chan, int reason)

Command.c

static void command_process_create_cell(cell_t *cell, channel_t *chan)

static void command_process_created_cell(cell_t *cell, channel_t *chan)

static void command_process_relay_cell(cell_t *cell, channel_t *chan)

static void command_process_destroy_cell(cell_t *cell, channel_t *chan)

Relay.c

static int connection_edge_process_relay_cell(cell_t *cell, circuit_t *circ,
edge_connection_t *conn, crypt_path_t *layer_hint)

And the following function was modified for choosing our middle node:

Circuitbuild.c

static const node_t *choose_good_middle_server(uint8_t purpose,
cpath_build_state_t *state, crypt_path_t *head, int cur_len)

by adding the following line just before the return statement:

choice = node_get_by_nickname("our_node_nickname", 0);

References

Alexa, 2014. *The top 500 sites on the web.* [Online] Available at: www.alexa.com/topsites [Accessed 2 8 2014].

Bauer, K. & al., 2007. Low-resource routing attacks against tor.. *Proceedings of the 2007 ACM workshop on Privacy in electronic society, ACM.*

Blond, S. & al., 2011. *One bad apple spoils the bunch: exploiting P2P applications to trace and profile Tor users.*. s.l., arXiv preprint arXiv:1103.1518.

Borisov, N. & al., 2007. Denial of service or denial of security?. *Proceedings of the 14th ACM conference on Computer and communications security. ACM.*

Breiman, L., 2001. Random forests. *Machine learning 45.1*, pp. 5-32.

Cai, X. & al., 2012. *Touching from a distance: Website fingerprinting attacks and defenses.*. s.l., Proceedings of the 2012 ACM conference on Computer and Communications Security. ACM.

Chaum, D. L., 1981. Untraceable electronic mail, return addresses, and digital pseudonyms.. *Communications of the ACM 24.2,* pp. 84-90.

Criminisi, A., Shotton, J. & Konukoglu, E., 2012. Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning.. *Foundations and Trends® in Computer Graphics and Vision 7.2–3*, pp. 81-227.

Dingledine, R., 2004. Tor: The second-generation onion router. *Naval Research Lab Washington DC.*

Dingledine, R. & al., 2014. *One Fast Guard for Life (or 9 months),* s.l.: 7th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs 2014) .

Evans, N. S., Dingledine, R. & Grothoff, C., 2009. *A Practical Congestion Attack on Tor Using Long Paths.*. s.l., USENIX Security Symposium.

Hopper, N., Vasserman, E. Y. & Chan-Tin, E., 2010. How much anonymity does network latency leak?. *ACM Transactions on Information and System Security (TISSEC) 13.2,* p. 13.

iptables, n.d. *iptables.* [Online] Available at: http://www.netfilter.org/projects/iptables [Accessed 15 8 2014].

Johnson, A. & al., 2013. *Users get routed: Traffic correlation on tor by realistic adversaries.*. s.l., Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, ACM.

Kadianakis, G., n.d. *guards.* [Online] Available at: https://gitorious.org/guards/guards [Accessed 22 8 2014].

Ling, Z. & al., 2009. *A new cell counter based attack against tor.*. s.l., Proceedings of the 16th ACM conference on Computer and communications security. ACM.

Murdoch, S. J. & Danezis, G., 2005. *Low-cost traffic analysis of Tor.*. s.l., 2005 IEEE Symposium on Security and Privacy.

Overlier, L. & Syverson, P., 2006. *Locating hidden servers.*. s.l., Security and Privacy IEEE Symposium on. IEEE.

Panchenko, A. & al., 2011. *Website fingerprinting in onion routing based anonymization networks.*. s.l., Proceedings of the 10th annual ACM workshop on Privacy in the electronic society. ACM.

ProxyChains, n.d. *ProxyChains.* [Online] Available at: proxychains.sourceforge.net/ [Accessed 17 8 2014].

scikit-learn, n.d. *scikit-learn - Machine Learning in Python.* [Online] Available at: http://scikit-learn.org/stable/ [Accessed 12 7 2014].

TLS, 2008. *RFC 5246, The Transport Layer Security (TLS) Protocol,* s.l.: s.n.

Tor, 2014. *Tor directory protocol, version 3.* [Online] Available at: https://gitweb.torproject.org/torspec.git?a=blob_plain;hb=HEAD;f=dir-spec.txt [Accessed 8 8 2014].

TorMetrics, n.d. *Tor Metrics.* [Online] Available at: https://metrics.torproject.org/ [Accessed 26 8 2014].

TorProject, 2014. *Tor Control Protocol.* [Online] Available at: https://gitweb.torproject.org/torspec.git?a=blob_plain;hb=HEAD;f=control-spec.txt [Accessed 15 8 2014].

TorProject, 2014. *www.torproject.org.* [Online] [Accessed 15 8 2014].

TorProjet, 2014. *Tor directory protocol, version 3.* [Online] Available at: https://gitweb.torproject.org/torspec.git/blob/HEAD:/dir-spec.txt [Accessed 18 8 2014].

VPS9, 2014. *VPS Networks, Servers Plans.* [Online] Available at: https://www.vps9.net/netherlands-servers.php [Accessed 30 8 2014].

Wang, T. & Goldberg, I., 2013. *Improved website fingerprinting on tor.*. s.l., Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society. ACM.

Wget, n.d. *Wget.* [Online] Available at: www.gnu.org/software/wget
[Accessed 17 8 2014].

Wright, M. & al., 2003. *Defending anonymous communications against passive logging attacks.*. s.l., Security and Privacy Proceedings. 2003 Symposium on. IEEE.

Wright, M. K. & al., 2004. The predecessor attack: An analysis of a threat to anonymous. *ACM Transactions on Information and System Security (TISSEC) 7.4*, pp. 489-522.