Detection of P2P and anonymity networks

Ondřej Fikar

December 2014

advisor: Ing. Ján Jusko

Czech Technical University in Prague
Faculty of Electrical Engineering, Department of Control Engineering

Acknowledgement

Let me thank my advisor Ing. Ján Jusko, my parents, and Kateřina.

Declaration

Abstract

V této práci navrhujeme metodu pro detekci Toru v počítačových sítích. S použitím klasických metod strojového učení, například SVM, se nepodařilo najít příznaky, které by dokázaly popsat Tor s dostatečnou přesností, a získané výsledky obsahovaly příliš mnoho falešně pozitivních výsledků. Provedli jsme rozbor společných vlastností anonymizačních nástrojů, abychom našli nějaké nestandardní příznaky, které by mohly sloužit k jejich identifikaci. Dospěli jsme k závěru, že stroje, které jsou součástí Toru, nebo případně jiného anonymizačního nástroje, mohou být spojeny na základě velkého množství svých sdílených kontaktů. Proto jsme se rozhodli použít přístupy teorie grafů a doplnili jsme původní klasifikační algoritmus detekcí komunit. Navrhovaný postup jsme testovali na datech z reálné sítě a zjistili jsme, že metoda je funkční a je schopná nalézt dostatečně velkou část strojů v dané síti, které jsou součástí Toru. Zároveň nezpůsobuje velké množství falešných poplachů. Přehled vybraných anonymizačních nástrojů a detailní popis Toru jsou rovněž součástí práce. V práci také shrnujeme relevantní poznatky z oborů strojového učení a teorie grafů.

Klíčová slova

Tor; anonymizační sítě; teorie grafů; detekce komunit; strojové učení

Abstract

In this thesis we propose a method for detection of Tor traffic inside computer networks. Traditional machine learning approaches, for example the SVM classifier, are not able to find features distinctive enough to identify Tor and the obtained results contain a large number of false positives. We analyse common traits of anonymity tools to find non-standard features which could be used for their identification and conclude that hosts participating in Tor and potentially other anonymity networks may be linked on the basis of a high number of their mutual contacts. Thus we employ graph theory and complement the original classification algorithm with community discovery. We evaluate the method on real network data and find it is able to identify hosts serving as Tor relays with high precision and acceptable recall. The analysis of Tor together with a survey of other anonymity tools is also included in the thesis. The thesis also contains a summary of relevant aspects of machine learning and graph theory.

Keywords

Tor; anonymity networks; graph theory; community detection; machine learning

Contents

Abbreviations

CaC	Command and control, a controlling structure of a botnet.
DHT	Distributed hash table, a data structure in peer-to-peer applications.
DoS	Denial of service, an attack based on exhaustion of resources.
IP	Internet protocol or Internet protocol address.
P2P	Peer-to-peer, a networking paradigm, where hosts may act both as a server and a client.
SVM	Support vector machine, a machine learning algorithm.

1 Introduction

In this text we propose a method to detect the Tor anonymity network which combines a traditional machine learning approach with graph based methods.

Anonymity tools are designed to provide privacy to their users by hiding with whom the users communicate and protecting the content of the communication from being read by third parties. From different point of view, anonymity tools conceal the originator of published information. But anonymity tools usually do not aim to hide the sole fact that the user is engaged in communication or that she is using tools to keep the communication private. Tor is, to our knowledge, the most popular anonymity network nowadays.

Our goal is to detect the use of Tor inside computer networks, especially in the corporate environment. By detection we mean identifying the hosts inside the given computer network which communicate via Tor. We do not want by any means to compromise the privacy of the users by linking them to their communication counterparts or the actions they perform on the Internet through Tor, nor do we want to analyze or decrypt the content of their messages.

We consider private use of anonymization tools completely legitimate. However, such behavior inside a corporate network often indicates potential or actual information breach. For example, malicious software has been reported to use Tor for its signaling[1]. Another reason to study anonymity tools is that certain malware utilizes decentralized command and control structures, which may have similar properties as the anonymity tools. The methods developed for anonymity tools can, therefore, help us to detect malware communication as well.

Anonymity tools usually utilize a group of hosts, called (depending on the terminology) relays, routers, or mixes, to repeatedly forward traffic generated by users before it is handed over to its intended destination. Many of these tools, including Tor, require users to connect to a relatively high number of relays in order to maintain the network structure, to serve as a relay for other users, or to prevent traffic analyses (see chapter 2). We believe that these connections form a notable community structure amongst the involved hosts.

Hence we propose a method, which constructs a graph representation of communication inside a given network and clusters the graph to reveal communities, especially the ones which comprise of Tor hosts. The method includes the following steps:

- Using a common machine learning approach we filter out the majority of hosts we are sure are not members of the Tor network.
- We construct a graph of mutual communication of the remaining hosts and cluster the graph to reveal significant groups.
- From the revealed clusters we identify the one which contains the Tor hosts by leveraging results of the first step.

The above-mentioned method is described in chapter 5. The rest of the Thesis is organized as follows: In chapter 2 we provide an overview of a number of anonymity tools with a detailed description of Tor. Chapter 3 introduces machine learning methods which are a part of our proposed method. In chapter 4 we look at the aspects of graph theory related to community discovery. In chapter 6 we discuss other detection methods based on graph theory we have encountered.

2 Anonymity networks

In this chapter we provide a survey of anonymity networks. To our knowledge Tor is the most popular anonymity tool these days and also detection of Tor is the objective of the method proposed in this text so we start the chapter with a detailed description of Tor in section 2.1. Then we briefly introduce other selected anonymity networks in section 2.2 and we conclude the chapter with a discussion of their common properties.

2.1 Tor

Tor is a circuit based low latency anonymous communication service. Its design is described in [2] and information on later changes in the protocol were published in [3] and [4]. All the information in this section is adopted from those sources, if not explicitly stated otherwise.

The anonymity provided by Tor is based on hiding the link between the user and her actions on the Internet. The main principle of its operation is that it relays the traffic from an originator of the traffic to its recipient through a sequence of routers. In Tor terminology these sequences are called **circuits**. Each router in the circuit knows only its predecessor and its successor and no one in the circuit (apart from the originator) knows both endpoints of the communication.

To prevent the routers and third parties from reading the content of the relayed messages (and thus identifying both of the endpoints from the routing information) the content and the additional routing informations are wrapped in multiple layers of encryption. As the traffic passes through the circuit each router along the path removes one layer of encryption. Finally the last router in the circuit removes the last layer and sends the original message to its destination. This arrangement is also known as the onion routing.

Tor consists of two parts. The software which runs on a user machine called an **onion proxy** and the **onion routers** which form the Tor network itself.

An onion proxy intercepts TCP streams produced by the software on a user machine during actions such as web browsing or IM messaging and routes them through the Tor network. The proxy takes care of all the communication with the routers and coordinates all the actions which are necessary for successful routing. A user does not have to understand what is going on under the hood. The onion routers are hosts running the same software with relaying capability enabled. The reason why Tor is capable of relaying only TCP streams comes from design trade-off between universality and usability. Operating on lower network layers allows Tor to handle variety of applications without including application specific features into the software. On the other hand, to enable Tor to work on even lower level protocols would require kernel modification on some systems.

2.1.1 Cells

The messages generated by an user flow through the circuits divided into units called **cells**, which, in a sense, play the same role as packets in lower level protocols. There are two types of a cell - command cells and relay cells. The former are used for signaling between the onion proxy and the routers and are always interpreted in the receiving node. The later carry the content intended for the final recipient and are always forwarded.

Each cell is 512 bytes long and consists of a header and a payload. A header of a cell contains routing information and a command for the receiving router. A payload of a command cell may contain additional information if needed for the execution of the command the cell carries. In case of relay cells the command is always *relay*. A payload of a relay cell contains additional relay header with information such as the message checksum and the actual relayed content. The whole payload of a relay cell is encrypted according to the onion scheme described in the introduction of this chapter.

2.1.2 Circuits

The pairs of routers adjacent in the circuit keep a TLS connection open to pass the traffic through. The establishment of the circuits and the TLS tunnel is a time consuming operation due to the asymmetric cryptography involved. To spare resources the circuit may carry multiple TCP streams and the TLS tunnel may multiplex many circuits (not necessarily originating from the same onion proxy).

To distinguish into which circuit the cell belongs each onion router keeps a list of circuits it participates in with their corresponding numbers. These numbers are connection specific, which means that the number is shared only between two directly adjacent nodes. A router keeps one number for each side of the circuit. When a cell arrives to a router, a circuit number from the cells header is used to to identify to which circuit the incoming cell belongs. Before forwarding to the next relay, the cells' circuit number is changed to the circuit number the router shares with the next node.

TCP stream relayed by one circuit may be possibly linked together because they reach the destination from the same router. To limit the number of linkable streams the circuits are rotated regularly. After a given time interval a circuit is considered expired and no new streams are relayed through it.[1] Once all streams in an the expired circuit are closed, the circuit is torn down.

2.1.3 Cell relaying

An onion proxy keeps a separate key for each router in a circuit and encrypt the cells, which are to be sent through the circuit by each of these keys. When a cell arrives to an onion router the command field in its header is checked. The command cells are interpreted in the given router. The relay cells are processed as follows: The payload of the cell is decrypted by the key corresponding to the circuit it came from and the router checks whether the checksum included in the cell corresponds to its content. If it does, it means the cell is completely decrypted and the relay is the last hop in the sequence. In that case the decrypted content of the cell is forwarded to the final destination according to the information in its relay header. If the checksum does not match the payload, the router changes the cell's header as described in section 2.1.2 and sends it to the next router in the circuit.

Relaying of the traffic in the opposite direction - from a destination host to an onion proxy - is done in the same way. The only difference is that the routers are adding the layers of encryption instead of removing them and the proxy has to decrypt all of them.

[1] According to [2] the default expiration interval is one minute. Our observation suggest that the actual value is close to 3 minutes for most of the clients in the wild.

2.1.4 Circuit construction

In this section we describe the process of circuit creation in detain. To make the explanation clearer, we denote the involved parties as follows: A stands for the onion proxy initiating the communication, B is the destination of the traffic, and R_i denotes the routers in the circuit with R_i being the first and R_n being the last. Usually n equals to 3.

A circuit is built iteratively. When a new one is to be open A chooses n routers from the list of know routers obtained from the Tor network(see section 2.1.5). Then A opens a TLS connection to R_1 and sends a *create* command through it with a chosen circuit number and its half of the information necessary to perform the Diffie-Hellman key exchange procedure. R_1 associates the received number with the newly established circuit and responds with its part of key exchange. Now A and R_1 share a key and the circuit between them is established.

To extend the circuit from R_i to R_{i+1}, A sends a relay cell with its part of the key exchange procedure to the R_i. Note, that the circuit up to R_i is already established so it may me used to transfer the information as described in section 2.1.3. Upon receiving the extend request R_i chooses a circuit number, opens a TLS connection to R_{i+1}, and use it to send R_{i+1} a *create* command with the selected circuit number and the handshake information received from A. R_{i+1} responds in the same manner as R_1 in the previous paragraph and the necessary information are transferred to A through the already existing part of the circuit. This step is repeated until the circuit reaches its intended length.

When R_n is joined, A asks it to open a TCP connection to B and the communication between the endpoints may begin.

As we already mentioned, the creation of a circuit takes notable time. To prevent latency, a number of new circuits is built in advance so there is always a circuit ready for use.

2.1.5 Directory

Tor directory provides information about the state of the Tor network. In particular, it contains information about the onion routers - their addresses, public keys, exit policies and other details. It is build collaboratively by well known servers and provided to the onion proxies via HTTP protocol.

When an onion router wants to join the Tor network for the first time, it has to ask the directory authorities to be listed in the directory. Administrator of each directory server has to confirm each router and add it to the directory manually. This way it is more difficult for an attacker to introduce a significant amount of malicious routers to the network.

The consensus document, which is the final directory version distributed to the proxies is a result of a negotiation of all of the authority servers. There are two main reasons for this. First, malicious routers has to deceive a number of independent parties to get through, which makes this kind of attack harder. Second, it is necessary that each onion proxy has the same information about the network. Otherwise various attacks based on a different knowledge may be feasible[5].

The consensus is built by a majority vote of all authoritative servers and is provided by the directory servers via HTTP. Usually it is obtained through the Tor network, so it is not so obvious the user is using Tor. In order to prevent overloading of the directory servers, the directory is cached by the onion routers, so direct request to the directory servers are not necessary any more. To prevent malicious routers to forge it, the consensus file is signed by the participating authorities.

To ensure the positive effect of the multiple collaborating authorities on the network resilience, they should be ran by independent parties. Those are individuals and different organizations, which are distributed around the globe in different jurisdictions.

2.1.6 Router selection

The routers to form a new circuit were originally selected with uniform probability. It turned out, that this approach caused bandwidth bottlenecks, so the algorithm was changed so that the probability of a router selection is based on the bandwidth the router provide.

Also, not every router is suitable for every position in the circuit. Only trusted relays are chosen as the first nodes in a circuit because this position is considered particularly critical for the user anonymity. Similarly, not every relay is suitable for the last position in a circuit because the exit policy of the relay has to allow the traffic, which is to be send through the circuit. A relay owner may even prohibit any traffic to leave Tor network through her relay.

The bandwidth, the exit policy, and the flag signaling whether the relay shall be used as an entry node of a circuit are published in the directory document. An onion proxy then weights all of this data when building a new circuit. Consequently, the probability of selection is not the same for every onion router.

2.1.7 Known attacks

In this section we list some known attack against the Tor network. The list is by no means exhaustive.

End to end correlation

Variety of attacks is possible, in case the attacker can observe both endpoints of the communication.

Correlation of patterns in time and volume of the traffic produced by the user and received by the recipient will eventually lead to traffic confirmation. Moreover, it is easy to tag the communication either by altering a timing of the packets or even disrupting completely at one side of the channel and observe, whether the communication stream on the other side is affected.

The Tor design considers local adversary and consequently focuses on preventing traffic confirmation. This means the case when an attacker already suspect a user to communicate with a given recipient and taps the endpoints to confirm the hypothesis. To discover a communication counterpart of a user without additional prior knowledge should not be feasible since the attacker would need visibility to a large portion of the Internet.

The adversary model of Tor explicitly does not consider an attacker with global visibility for two reasons. First, the designers of Tor network believe that the large scale observation would be to difficult and expensive. Second, it is difficult if not impossible to retain low latency when designing service resilient to such kind of adversary.

A user may reduce the threat of the end to end correlation by running an onion router alongside her onion proxy. This way it will be more difficult for an attacker to distinguish the traffic produced by the user from the traffic being relayed.

Poisoning the network

An adversary may introduce malicious routers to the network with hope that a user will choose some of these routers as the first and the last hop in the circuit. This would allow end to end correlation in the same way as tapping the wire near the endpoints. According to [2], by introducing n malicious routers to the network of N nodes the adversary is able to observe at most $(\frac{n}{N})^2$ of the traffic. The fraction of observed traffic may be even increased by providing unusually high bandwidth and setting permissive exit policy.

To lower the risk of observation guard nodes were introduced to the Tor design. With guards enabled the onion proxy does not choose the first node in a circuit at random from all the routers available. Instead it keeps a list of already used routers and when new circuit is to be built, the first node is drawn from the list. New routers are used only when there is no other available router in the list.

Interception of plain-text traffic

The traffic leaves the exit node in the same form as it was intercepted by the onion proxy. This means in plain text, if it was originally unencrypted. A malicious exit node may analyze the outgoing traffic which could lead to immediate deanonymization of the users, if some sensitive information was present in the traffic. For example, there are users, who use Torrent over Tor for downloading pirated content. Unfortunately for them, Torrent protocol reveals the IP addresses of its users in order to build its overlay network. This fact makes the anonymity features of Tor useless[6].

Even if no sensitive information is present another attack is still possible. The adversary may alter a content, which is not protected by end-to-end authentication. This may have various consequences. It has been observed that a malicious exit node appended a malware to binaries, which were downloaded via Tor[7]. The reader may imagine even more subtle changes of the transferred information, which could cause serious trouble to the users. Of course employing of well known authentication methods would mitigate this kind of attack.

Iterated compromise

An attacker may try to link hops of the circuit from the recipient to the user. This may be done by various methods including exploiting unknown vulnerabilities in Tor software or starting a legal action against the owners of the corresponding routers. In any case, this has to be done fast since Tor provides perfect forward secrecy and once the encryption keys are discarded, there is no easy way to decrypt the communication.

Denial of service

Various DoS attacks are possible in Tor. For example, an attacker can force onion router to perform computationally intensive operation by extensive circuit creation. Another possible way to carry out DoS attack is to transmit dummy traffic through routers in order to render them unusable to benign users. This may, for example, attract more users to compromised exit nodes.

Exit node abuse

Malicious users may abuse the privacy provided by the Tor network to avoid prosecution for performing action which are considered illegal or antisocial. While this is not an actual attack against Tor, it may cause difficulties to exit node owners and possible exit node shutdowns. As such, it may negatively affect the whole network.

Tor designers point out that the cybercriminals already possess means to hide their actions, which are frequently more efficient than Tor. Consequently, the abuse of Tor should be rare.

Also, Tor may be used by users which are not considered cybercriminals in the usual sense, such as Torrent users. We already mentioned that the users, which are tunneling Torrent over Tor do not enjoy the same level of privacy as the other users but such behavior may attract unwanted attention to the routers anyway.

PR attacks

This class of attacks uses similar methods as the exit node abuse attacks. The difference between the two is that the purpose of the PR attacks is to harm the public image of Tor and thus discourage users from using it. The lower is the number of users the easier is to deanonymize them[8].

2.2 Other anonymity networks

Mixminion[5] is a remailer with single use reply blocks. It provides anonymity to user emails by resending them through a series of relays. Each of the relays provides public key through a dedicated directory server the messages to be sent are encrypted by those keys and the encryption is removed as the message passes the relays. The path of the message is arbitrary and is up to the user to decide, which relays to use. The messages contain a reply blocks, which may be used to send a reply to the originator of the message without compromising her identity. The reply block are single use only but the network provides a method to maintain long term pseudonyms based on these reply blocks. In order to avoid correlation attacks the messages are batched in the relays and padded, also dummy traffic is produced for further obfuscation. To prevent abuse the relays may set exit policies to filter emails leaving the network.

Crowds[9] and **Hordes**[10] are related anonymity networks designed for web browsing. Crowds routes user requests through a series of relays here called *jondos* (pronounced as John Doe). Each user of the network serves as a jondo as well. This way no one knows whether a traffic arriving from a jondo is generated by its owner or is just relayed. Also the other users benefit from the provided bandwidth. The operation of the network is controlled by a central server called a *blender*, which distributes the public keys of jondos. The routing path through the network last about 24 hours and users (including newcomers) are not allowed to create a path at will. All users reroute at once when a signal is sent from the blender. This is to prevent identification of users based on path creation. Smaller private *crowds*, groups of jondos, may be created to reduce the communication overhead and to group trusted users. A user needs an account held by the central authority.

Hordes improve the Crowds by utilizing multi-cast groups for distributing the replies to the users. Each member of a given group receives the reply but if the group is large enough the anonymity of the receiver is not in danger.

I2P[11] is another example of a low latency anonymity network. It is a structured P2P network based on DHT similar to Kademlia. Unlike Tor, I2P is fully distributed with no central authority similar to Directory servers and it is not meant for communication outside the network (even though it is possible to use some of the relays as a proxy for such actions). The atomic unit of the protocol, which contains a user payload is called a cell. A number of cells are grouped to a garlic-like structure called a clove; cloves are transmitted through the members of the network which are called relays. When a relay receives a clove it splits it to the cells mix them with cells from other received cloves and repackage them to new cloves which are then transmitted further. Cloves are padded and their retransmitting in relays might be delayed. In order to communicate via I2P a user host open tunnels to other relays in the network. These tunnels are of two kinds, inbound and outbound and are strictly one directional. By default a user keeps 4 tunnels open (two of each kind) and change tunnels after 10 minutes of operation. Fast and reliable relays are preferred for tunnel creation. There exist many plug-ins for I2P allowing web publishing inside the network, chatting, file sharing etc.

Freenet[12] is a P2P network for anonymous content publishing and consuming. The network is fully decentralized with each node keeping its own dynamic routing records. Published files and the nodes of the network are identified by cryptographic keys. When a file is published a short text description is made for it; the description is hashed to obtain the file identifier. The keys for nodes are assigned collaboratively when a new host joins the network. When a user wish to obtain a file it sends a query containing the file identifier to its neighbors in the network. If a node receives a query for a file it does not possess it forwards the query to its neighbor which has the most similar key to a key of the requested file. When the file is located, it is sent back along the path of the query and each node along the path keeps a copy of it. In this setting it is infeasible identify the origin of the files distributed in the network. Each node knows only about its own neighbors and makes local decisions about the routing. The mechanism should ensure that the nodes will *specialize* for storing the files with similar keys. Similarity of keys does not imply similarity of topics so the network as a whole is resilient to departures of individual nodes.

Rumor Riding[13] is a P2P anonymity network with an interesting approach to the routing of messages. In order to reduce the overhead connected with the asymmetric cryptography and the path creation it propagates the messages in a random walks manner. When a user wish to send a message, she generates a symmetric key, which she uses to encrypt the message. Then she mark the key and the encrypted message with same random number and sends *both* the message and the key through the network, each to a different node. When a node receives a message or a key it forwards it to a randomly chosen peer unless it already possesses the counterpart for the received object. If so it uses the key to decrypt the message and then forwards the message to its destination without knowing the originator of the message.

Cashmere[14] is a structured P2P anonymity network, which uses regions in namespace to replace individual relays. This design decision improves the network resiliency to node churn. The nodes in Cashmere are given random identifiers and are grouped according to the prefixes of their keys. The members of each group share one public-private key pair common for the whole group. Path of the message transmitted through the network does not comprise of individual relays but of the whole groups. Any member of the group, which receive the message is able to decrypt the routing information and to forward it to another group along the path. The partially decrypted payload is also send to all members of the group. The recipient is a member of the last group and this way it is able to receive the message without compromising its identity. The other members of the group are not able to remove the last layer of encryption because the recipient private key is needed to do so.

Generally, the principle of the anonymity networks we are aware of is to relay the user generated traffic through another host or a series of them hosts before it reaches its destination. The traffic is usually mixed with traffic originating from other users and sometimes its properties are altered before or during the transfer. Example of such alternation may be adding delays in relaying hosts or padding of the transmitted content. Also dummy traffic may be generated to further complicate the traffic analysis.

Padding and other normalization of the traffic may introduce significant patterns for traffic fingerprinting. Such patterns do not allow analysis of the content of the communication, as they are identical for all the generated traffic, but they may be used to identify, which anonymity tool is being used. Randomization of the traffic may provide similar information, in case the generated noise is unique to the deployed anonymity tool. The drawback of these features is that they are specific for the given anonymity network.

We consider the *social* aspect of anonymity tools more common for anonymity networks. Many anonymity networks, especially those based on P2P paradigm, requires the users to connect to relatively high number of relays. Our intuition is that not many other users will communicate with these relays and thus the relays should be linkable on the basis of common users and vice versa. A proper community discovery procedure may be able to separate those users or relays from the other hosts. This theory is the key aspect of the detection method we propose in this text.

On the contrary some anonymity networks tend to make stable connections to low number or even one relay. Our proposed method would not perform well, when applied to such networks. But the stability of the connections might be used as a distinctive feature to design another detection method.

3 Classification methods

In this chapter, we are going to review some machine learning methods we used in our proposed detection method in chapter 5. Namely, we will discuss Logistic regression and Support vector machine.

The classification problem is usually defined as follows: We are given a set of observations $x_1, \dots x_N$, where $x_i \in \mathbb{R}^n$ is a vector of values measured for one observed item. The elements of observation vectors are called features and the space from which are the vectors drawn is called a feature space. Each observation belongs to one of m classes encoded by numbers $1 \dots m$. For the N observations we know to which class they belong and we want to find a function f, which assigns each item to its class or estimate $P(G = k \mid X = x_i)$, the probability that the item represented by observation x_i belongs to class k. The function f is called a classifier. We also want the classifier to be able to correctly classify observations, which were not part of the original set $x_1 \dots x_N$. In the following sections we will assume that there are only two classes, to which may observations belong and we will encode them as {0, 1} or {-1, 1}, however he presented methods may be generalized to an arbitrary number of classes[15].

The following material on Logistic regression is adopted from [15] and [16], the material on Support vector machines is adopted from [17].

3.1 Logistic regression

In case of two class classification problem, we wish to find a suitable function to model the probability $P(G = 1 \mid X = x_i)$. One such a function may be a logistic function (also called a sigmoid) of a form

$$p(x;\beta) = \frac{e^{\beta^T x}}{1 + e^{\beta^T x}}, \tag{1}$$

where $x \in \mathbb{R}^{n+1}$ is the observation and $\beta \in \mathbb{R}^{n+1}$ is a parameter of the algorithm. In this section we will include an intercept parameter β_0 to β and for this reason, we extend x by adding a first element equal to 1 to it. Then $p(x;\beta)$ is a shorter notation for $P(G = 1 \mid X = x)$, which emphasize the role of parameter β. A sigmoid function has some desirable properties for modeling probability, such as its range is $(0, 1)$ and it is a monotone function of x.

From equation 1 and natural requirement for the probabilities to sum to 1,

$$P(G = 1 \mid X = x) + P(G = 0 \mid X = x) = 1, \tag{2}$$

it follows that logistic regression models logarithmic odds of two classes being linear. Odds of two events is a fraction of their probabilities, that is for events A and B the odds is $\frac{P(A)}{P(B)}$. In our case we are interested in odds of probabilities that observation x belongs to class 1 or to class 0, which is expressed as

$$\log \frac{P(G = 1 \mid X = x)}{P(G = 0 \mid X = x)} = \beta^T x. \tag{3}$$

To use logistic regression for classification we have to find optimal value of parameter β in a sence of maximal likelihood. The likelihood of parameter θ is defined as

$$l(\theta) = \sum_{i=1}^{N} \log p_{g_i}(x_i; \theta) \tag{4}$$

from which follows that the likelihood of β is

$$l(\beta) = \sum_{i=1}^{N} y_i \log p(x_i; \beta) + (1 - y_i) \log(1 - p(x_i; \beta)) = \sum_{i=1}^{N} y_i \beta^T x_i - \log(1 + e^{\beta^T x_i}), \tag{5}$$

where x_i is an input variable for observation i and y_i encodes the class to which the observation belongs.

To maximize $l(\beta)$ one may employ Newton-Raphson algorithm, which improve β iteratively:

$$\beta^{\text{new}} = \beta^{\text{old}} - \left(\frac{\partial^2 l(\beta)}{\partial\beta\partial\beta^T} \right)^{-1} \frac{\partial l(\beta)}{\partial\beta}. \tag{6}$$

The first derivative obtained form equation 5 is

$$\frac{\partial l(\beta)}{\partial\beta} = \sum_{i=1}^{N} y_i x_i - \frac{e^{\beta^T x_i}}{1 + e^{\beta^T x_i}} = \sum_{i=1}^{N} x_i (y_i - p(x_i; \beta)) \tag{7}$$

and the second derivative is

$$\frac{\partial^2 l(\beta)}{\partial\beta\partial\beta^T} = \sum_{i=1}^{N} -x_i^2 p(x_i; \beta)(1 - p(x_i; \beta)). \tag{8}$$

The update expression for β may be rewritten to a convenient matrix notation. Let $X \in \mathbb{R}^{N \times (n+1)}$ be the matrix of observations, such that its ith row is the observation vector x_i^T, let p be the vector of probabilities, such that for its ith element holds $p_i = p(x_i; \beta^{\text{old}})$, and let $W \in \mathbb{R}^{N \times N}$ be a diagonal matrix, such that

$$W_{ij} = \begin{cases} p(x_i; \beta^{\text{old}})(1 - p(x_i; \beta^{\text{old}})), & i = j \\ 0, & i \neq j. \end{cases} \tag{9}$$

Then we may rewrite the derivatives as

$$\frac{\partial l(\beta)}{\partial\beta} = X^T (y - p) \tag{10}$$

and

$$\frac{\partial^2 l(\beta)}{\partial\beta^2} = -X^T W X. \tag{11}$$

And finally the update equation as

$$\beta^{\text{new}} = (X^T W X)^{-1} X^T W z, \tag{12}$$

where $z = X\beta^{\text{old}} + W^{-1}(y - p)$.

Once is the optimal value of β computed, observation x may be classified by computing the corresponding value of $p(x; \beta)$ from equation 1. Usually the observation is classified as belonging to class 1 if $p(x; \beta) \geq 0.5$, but the threshold may be arbitrary. From equation 1 follows that condition $p(x; \beta) = t$, where $t \in (0, 1)$ is a given threshold is satisfied if $\beta^T x = \log\left(\frac{t}{1-t}\right)$. This equation represents a hyperplane in the feature space, which divides the observations which are classified to different classes. The hyperplane is usually called a decision boundary.

3.2 Support vector machine

Support vector machine (SVM) is a classification algorithm, which tries to find an optimal decision boundary between the classes. This is such a boundary which maximizes its distance to the closest observations. Such a distance is called a margin and we expect that the wider is the margin the lower is the risk of misclassifying the future observations. Additionally SVM allows for complex non-linear decision boundaries by employing a technique called a kernel trick.

3.2.1 Separable case

Suppose we have N observations $x_i \in \mathbb{R}^p$ belonging to two classes with known labels $t_i \in \{-1; 1\}$ and we wish to separate them by a linear classifier of the form

$$y(x) = w^T \phi(x) + b, \tag{13}$$

where $w \in \mathbb{R}^p$, $b \in \mathbb{R}$ are the parameters and $\phi(x)$ is a transformation of the original feature space. With such a classifier a label for observation x may be estimated by evaluating sign $y(x)$.

Also suppose the two classes are linearly separable, i.e. there exist a hyperplane which perfectly separate the observations of different classes. In such case we can find a classifier of the form 13 for which it would hold true that $t_i y(x_i) > 0$, for all i. If is the linear separation possible, there exist many decision boundaries which fulfill the task. We aim to find such a classifier which would place largest possible margin between the classes.

Having a vector x and a hyperplane defined by $y(x) = 0$ we may decompose x to $x = x_\perp + x_\parallel$, where $x_\perp$ is perpendicular to w and $x_\parallel$ is parallel to it. The perpendicular distance to the hyperplane is then given by the length of $x_\parallel$, which may be rewritten as $r\frac{w}{\|w\|}$, where r is its length. Hence we have

$$x = x_\perp + r\frac{w}{\|w\|}. \tag{14}$$

Multiplying the equation by w^T and adding w_0 we get

$$w^T x + w_0 = w^T x_\perp + w_0 + r\frac{w^2}{\|w\|} \tag{15}$$

and from $w^T x + w_0 = y(x)$ and $w^T x_\perp + w_0 = y(x_\perp) = 0$ follows

$$r = \frac{|y(x)|}{\|w\|}. \tag{16}$$

We replace $|y(x)|$ with $t_i y(x_i)$ and allow the feature space transformation $\phi(x)$ to get

$$r_i = \frac{t_i y(x_i)}{\|w\|} = \frac{t_i(w^T\phi(x) + b)}{\|w\|}. \tag{17}$$

The size of the margin is given by the distance of the closest observation to the decision boundary. Hence, we can maximize the size of the margin by maximizing

$$\arg\max_{w,b} \left(\frac{1}{\|w\|} \min_i [t_i(w^T\phi(x) + b)] \right) \tag{18}$$

with respect to w and b. Since rescaling w and b to κw and κb does not change the distance, the task can be made easier by setting $t_i(w^T\phi(x)+b) = 1$ for the closest point. Consequently it holds true that

$$t_i(w^T\phi(x) + b) \geq 1, \ \forall i = 1, \dots, N \tag{19}$$

and we only have to maximize $\frac{1}{\|w\|}$ or, equivalently, minimize $\|w\|^2$. So the task to solve now is

$$\arg\min_{w,b} \frac{1}{2}\|w\|^2 \tag{20}$$

subject to equation 19. Factor $\frac{1}{2}$ is included for later convenience. Introducing Lagrange multipliers $a_i \geq 0$ we may construct Lagrangian

$$L(w, b, a) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{N} a_i[t_i(w^T\phi(x) + b) - 1], \tag{21}$$

where a stands for the vector of the multipliers $(a_1, \dots, a_N)^T$. Setting the derivatives of L w.r.t. w and b to zero, we obtain

$$w = \sum_{i=1}^{N} a_i t_i \phi(x_i), \tag{22}$$

$$0 = \sum_{i=1}^{N} a_i t_i, \tag{23}$$

which can be used to eliminate w, b from equation 21 leading to the dual representation

$$\tilde{L} = \sum_{i=1}^{N} a_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} a_i a_j t_i t_j \mathrm{k}(x_i, x_j), \tag{24}$$

which is to be maximized with respect to a subject to

$$a_i \geq 0, \tag{25}$$

$$\sum_{i=1}^{N} a_i t_i = 0. \tag{26}$$

Here we replaced the dot product $\phi(x_i)^T\phi(x_j)$ by a kernel function $\mathrm{k}(x_i, x_j)$, which allows us to employ different kernels. The kernels allow feasible feature space transformation but are out of scope of this text. Detailed description of theory related to kernels may be found in [15, 17].

Maximization of $\tilde{L}$ is a quadratic programming problem and its solution may be found in the literature. The classification of an observation may be computed in the dual form by using equation 22 to obtain

$$y(x) = \sum_{i=1}^{N} a_i t_i \mathrm{k}(x, x_i) + b. \tag{27}$$

It may be shown that the optimization satisfies Karush-Kuhn-Tucker conditions

$$a_i \geq 0 \tag{28}$$

$$t_i y(x_i) - 1 \geq 0 \tag{29}$$

$$a_i(t_i y(x_i) - 1) = 0, \tag{30}$$

from which follows that either $a_i = 0$ or $t_i y(x_0) = 1$. The observations with multipliers equal to zero will not appear in equation 27. We scaled w in such way that $t_i y(x_0) = 1$ holds true for observations, which are closest to the decision boundary, i.e. which lie exactly on the boundary of the margin separating the two classes. This means that only the observation which lie exactly on the border of the margin play role in classification. These observations are called support vectors. This fact allows us to solve for parameter b because

$$t_i y(x_i) = t_i \sum_{j \in S} a_j t_j \mathrm{k}(x_i, x_j) + b = 1, \tag{31}$$

where S is the set of support vectors. To compute the value of b we may use any support vector but from numerical point of view it is better to use all of them and then average the obtained values.

3.2.2 Inseparable case

When the observations are not linearly separable we introduce slack variables $\xi_i \geq 0$, such that $\xi_i = 0$ if the observation is outside the margin or on its boundary, and $\xi_i = |t_i - y(x_i)|$ otherwise. Consequently, $\xi_i = 1$ if the observation lies on the decision boundary and $\xi_i > 1$ if it is misclassified. The classification constraints 19 are then replaced by

$$t_i y(x_i) \geq 1 - \xi_i. \tag{32}$$

This is referred as relaxing the margin constraints to soft margin constraints. Now the goal is to minimize

$$C \sum_{i=1}^{N} \xi_i + \frac{1}{2}\|w\|^2, \tag{33}$$

where $C > 0$ is a parameter. Introducing Lagrange multipliers $a_i, \mu_i \geq 0$ we construct Lagrangian

$$L(w, b, \xi, a, \mu) = \frac{1}{2}\|w\|^2 + \sum_{i=1}^{N} xi_i - \sum_{i=1}^{N} a_i[t_i y(x_i) - 1 + xi_i] - \sum_{i=1}^{N} \mu_i \xi_i, \tag{34}$$

for which the corresponding Karush-Kuhn-Tucker conditions are

$$a_i \geq 1, \tag{35}$$

$$t_i y(x_i) - 1 + \xi_i \geq 0, \tag{36}$$

$$xi_i \geq 0, \tag{37}$$

$$\xi_i \mu_i = 0. \tag{38}$$

By setting the derivatives of L to zero we obtain

$$\frac{\partial L}{\partial w} = 0 \implies w = \sum_{i=1}^{N} a_i t_i \phi(x_i), \tag{39}$$

$$\frac{\partial L}{\partial b} = 0 \implies \sum_{i=1}^{N} a_i t_i = 0, \tag{40}$$

$$\frac{\partial L}{\partial \xi_i} = 0 \implies a_i = C - \mu_i. \tag{41}$$

Substituting the above equations to L we get the dual formulation

$$\tilde{L} = a_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j t_i t_j \mathrm{k}(x_i, x_j), \tag{42}$$

which we maximize with respect to a_i subject to

$$0 \leq a_i \leq C, \tag{43}$$

$$\sum_{i=1}^{N} a_i t_i = 0. \tag{44}$$

The quadratic optimization problem is again to be solved using standard techniques from the literature. Observations are classified in the same way as in separable case, i.e. by evaluating equation 27. Again, $a_i = 0$ do not contribute to predictions and the remaining of the observations are the support vectors. Parameter b may be computed from

$$t_i \left(\sum_{j \in S} a_j t_j \mathrm{k}(x_i, x_j) + b \right). \tag{45}$$

4 Graph theory

In this section we provide a short introduction to graph theory with focus on graph clustering, especially the algorithms used in the detection method we propose in chapter 5.

A graph is a mathematical model of bilateral relationships. The relationships may be binary (existing or non existing) or they may be specified by a numerical value, usually from $\mathbb{N}$ or $\mathbb{R}$.

Formally a graph G is a pair of sets $G = (V, E)$, where V is a set of vertices or nodes of the graph and $E \subset V \times V$ is a set of unordered pairs of vertices called the edges. If $(a, b) \in E$ we say that the vertices a and b are connected with an edge or that they are incident. We may also say that the edge $e = (a, b)$ is incident to the vertices a and b and vice versa. We usually visualize a graph as a set of points representing its vertices connected with lines representing its edges. An example of such visualization is in figure 6.

We may enhance the graph definition to $G = (V, E, w)$, where $w : E \to \mathbb{R}$ (or $w : E \to \mathbb{N}$) is a function assigning weights to the edges. A graph with weights assigned to its edges is called a weighted graph. If the pairs in the set E are ordered we call the graph directed. In this text we discuss only undirected graphs. If there are sets $V_1, V_2 \in V$, such that $V_1 \cap V_2 = \emptyset$ and $E = \{(a, b) \mid a \in V_1, b \in V_2\}$, we call the graph bipartite with partitions V_1, V_2. Notion of bipartite graph may be generalized to n-partite in a straightforward manner.

Given a vertex $x \in V$, the set $\{v \in V \mid (x, v) \in E\}$ is called a neighborhood of x and the vertices within the set are called the neighbors of x. We call the size of the neighborhood of the vertex x the degree of x and denote it as $d(x)$. Some authors

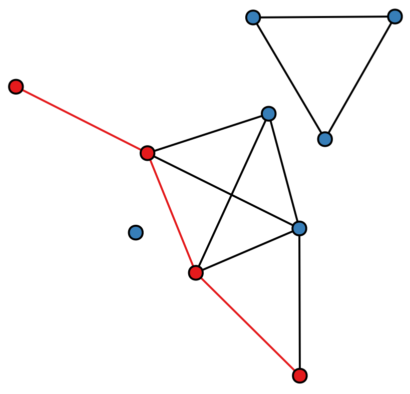


Figure 1 An example of a graph with ten vertices split into three component and a path (in red). *(Colors from www.ColorBrewer.org by Cynthia A. Brewer, Geography, Pennsylvania State University.)*

define a degree of a vertex in a weighted graph as the sum of weights of the edges which originate in the vertex, i.e. $d(x) = \sum_{(x,v)\in E} w(x, v)$.

A graph $G_1 = (V_1, E_1)$, such that $V_1 \subset V$, $E_1 = \{(a, b) \in E \mid a, b \in V_1\}$ is called a subgraph of G. We denote the relationship of being a subgraph by $G_1 \subset G$. A path in a graph is a series of vertices $\{x_1, \dots x_n\}$, such that $x_i \in V$ for all $i = 1, \dots n$ and $(x_j, x_{j+1}) \in E$ for all $j = 1, \dots n - 1$. We say that vertices a, b are connected if there exist a path in the graph, which starts in a and ends in b. A set of vertices is connected if each pair of the vertices from the set is connected. A maximal connected subgraph of a graph is called a component or a connected component. Many real world graphs, which are split to multiple components contain one component, which is substantially larger then the other ones. We call such a component a giant component.

A useful way to describe a graph is an adjacency matrix. If we order a set of vertices of a graph into a sequence $\{x_1, \dots x_N\}, N = |V|$, an adjacency matrix is a matrix $A \in \mathbb{R}^{N\times N}$ with elements such that

$$A_{ij} = \begin{cases} 1, & (x_j, x_i) \in E \\ 0, & \text{otherwise.} \end{cases} \tag{46}$$

For weighted graphs the adjacency matrix is denoted by W and contains the weights of the edges of the graph:

$$W_{ij} = \begin{cases} w(x_j, x_i), & (x_j, x_i) \in E \\ 0, & \text{otherwise.} \end{cases} \tag{47}$$

The number of edges in a graph may be computed as $\frac{1}{2}\sum_{x\in V} d(x)$. The sum is divided by 2 because we count each edge for both of its incident vertices. Having a graph with N vertices, the maximal possible number of edges in a graph is $\frac{1}{2}N(N-1)$.

Graph theory constitutes a useful tool for modeling many real world phenomena, such as social structures, computer networks, power grids, protein interactions and many others. Exhaustive coverage of the classical graph theory may be found in [18]. A friendly introduction to the more recent aspects of the graph theory, including random graphs and complex networks, may be found in [19].

4.1 Graph clustering

To cluster a graph means to divide it into communities of vertices. What exactly a community is depends heavily on the application field or the author. No generally accepted definition exists. Usually a cluster of vertices is required to be connected and the number of inter cluster connections is required to be significantly higher than the number of connections which link different clusters of the graph. This idea may be formalized as follows:

Let $G = (V, E)$ be a graph and C its subgraph. The average density of edges in G, $\delta(G)$, may be calculated as

$$\delta(G) = \frac{|E|}{\frac{1}{2}N(N-1)} \tag{48}$$

Expressing the number of internal and external edges of the subgraph C as

$$d_{\text{int}}(C) = |\{(x_1, x_2) \in E | x_1, x_2 \in C\}| \tag{49}$$

and

$$d_{\text{ext}}(C) = |\{(x_1, x_2) \in E | x_1 \in C, x_2 \notin C\}|, \tag{50}$$

respectively, one may define intra-cluster density $\delta_{\mathrm{int}}(C)$ and inter-cluster density $\delta_{\mathrm{ext}}(C)$ of subgraph C as

$$\delta_{\mathrm{int}}(C) = \frac{d_{\mathrm{int}}(C)}{\frac{1}{2}N_c(N_c-1)} \tag{51}$$

$$\delta_{\mathrm{ext}}(C) = \frac{d_{\mathrm{ext}}(C)}{\frac{1}{2}N_c(N-N_C)}, \tag{52}$$

where N_C and N denotes the number of vertices of C and G, respectively. For C to be a cluster of vertices in G we expect $\delta_{\mathrm{ext}}(C) \ll \delta(G) \ll \delta_{\mathrm{int}}(C)$[20].

A discussion of desirable cluster properties and a survey on clustering algorithms may be found in [20] and [21]. We are going to describe selected clustering algorithms in the following sections.

4.1.1 Spectral clustering

By spectral clustering we mean a family of clustering algorithms which make use of eigenvalues and eigenvectors of a matrix obtained from the graph. A natural choice would be the adjacency matrix or the matrix of the graph weights. It turns out, we may obtain better results when using a matrix called graph Laplacian though[20]. We will discuss Laplacian in the next section.

Laplacian

Imagine there is a substance distributed over the vertices of a graph. ψ_i denotes the amount of the substance in the vertex x_i. The substance may flow between the vertices and we assume that the rate of the transfer of the substance from vertex x_j to vertex x_i is $c(\psi_j - \psi_i)$, where c is a positive constant, if there is an edge between the corresponding vertices. The rate of change of ψ_i is proportional to $\psi_j - \psi_i$. Such process is called diffusion. Diffusion is described by the following equation

$$\frac{\mathrm{d}\psi_i}{\mathrm{d}t} = c\sum_j A_{ij}(\psi_j - \psi_i). \tag{53}$$

Equation 53 may be rewritten such that

$$\frac{\mathrm{d}\psi_i}{\mathrm{d}t} = c\sum_j A_{ij}\psi_j - c\psi_i\sum_j A_{ij} = c\sum_j A_{ij}\psi_j - c\psi_i k_i = c\sum_j (A_{ij} - \delta_{ij}k_i)\psi_j, \tag{54}$$

which in a matrix notation is

$$\frac{\mathrm{d}\psi_i}{\mathrm{d}t} = c(A-D)\psi, \tag{55}$$

where ψ is a vector composed of ψ_i and D is a diagonal matrix of vertex degrees. The last equation gives formula for the graph Laplacian

$$L = D - A \tag{56}$$

so that equation 55 may be rewritten to

$$\frac{\mathrm{d}\psi_i}{\mathrm{d}t} + cL\psi = 0, \tag{57}$$

which is the same form as a diffusion equation for gas with Laplacian in place of the Laplace operator [19]. In case of a weighted graph, matrix A and W may be used interchangeably assuming the items of W are non-negative[22]. In the following text we will suppose $L = D - W$.

A graph Laplacian has interesting properties related to clustering. Namely the number of its eigenvalues equal to zero is equal to the number of component of the graph. To see this we examine two theorems from [22]:

Theorem 1. *Matrix L satisfies the following properties:*

1. *For every vector $f \in (R)^n$ we have*
$$f^T L f = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} (f_i - f_j)^2. \tag{58}$$
2. *L is symmetric and positive semi-definite.*
3. *The smallest eigenvalue of L is 0, the corresponding eigenvector is the constant vector $c\mathbb{1}$, where c is a constant.*
4. *L has non-negative, real-valued eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$.*

Theorem 2. *Let G be an undirected graph with non-negative weights. The multiplicity k of the eigenvalue 0 of L equals the number of connected components $A_1, \ldots A_k$ in the graph. The eigenspace of eigenvalue 0 is spanned by the indicator vectors $\mathbb{1}_{A_1}, \ldots \mathbb{1}_{A_n}$ of those components.*

The indicator vector of a component is a vector with 1's in positions of the vertices, which belong to the component and 0's elsewhere.

Proof. We start with the case $k = 1$. Assume that f is an eigenvector with eigenvalue 0. We know that
$$0 = f^T L f = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} (f_i - f_j)^2. \tag{59}$$
From the equation follows that if x_i and x_j are connected, and thus $w_{ij} > 0$, the corresponding entries of the eigenvector $f_i f_j$ must be equal. Since the graph comprises one component, all its vertices are connected and f_i is constant for all i.

Considering the case $k > 1$ we may assume without loss of generality that the vertices are ordered according to the component they belong to. In such a case the matrix W has a block diagonal form and consequently the same is true for L. Each block of L_i of the Laplacian L is a Laplacian of the connected subgraph represented by the block. As such it has an eigenvalue 0 with corresponding eigenvector being an indicator vector of the component A_i.

Spectrum of a block diagonal matrix is a union of spectra of its blocks, thus L has eigenvalue 0 with multiplicity k. The eigenvectors corresponding to 0 are the indicator vectors of components A_i. □

The definition of Laplacian derived from the diffusion process we discussed above is not the only one used in literature. There are two other versions of Laplacian discussed in [22]:
$$L_{\text{sym}} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} W D^{-1/2} \tag{60}$$
$$L_{\text{rw}} = D^{-1} L = I - D^{-1} W. \tag{61}$$
Those are called normalized Laplacians and the symbols are derived from the fact that L_{sym} is a symmetric matrix and L_{rw} is closely related to random walk. Both normalized Laplacians have properties similar to the unnormalized Laplacian. Paper [22] discusses three related spectral clustering algorithms based on these three definition of Laplacian.

Spectral clustering algorithm

We have demonstrated that when a graph is divided into separate components, we can easily infer to which component the vertices of the graph belong by examining the eigenvectors corresponding to the eigenvalue 0. Each of these eigenvector is an indicator vector for one component of the graph, i.e. its entries are some positive constant for vertices, which belong to the corresponding component and zero for the others. In other words these eigenvectors, when scaled so its nonzero elements equal to one, point to a vertex of a hypercube in $\mathbb{R}^N$ and each component correspond to a different vertex.

If the graph is not separated into components but it still contains clusters in a sense of equations 49 and 50, we may consider the graph to be a perturbation of a disconnected graph. In such case the adjacency matrix, the weight matrix, and the Laplacian will be of the form

$$\tilde{A} = A + H_A \tag{62}$$

$$\tilde{W} = W + H_W \tag{63}$$

$$\tilde{L} = L + H_L, \tag{64}$$

where the letter with tilde denotes the perturbed matrix, the letter without decoration denotes the original matrix and $H_\bullet$ denotes the corresponding perturbation[1]. From the matrix perturbation theory we know that differences between the eigenvectors of the original matrices and the perturbed matrices are bounded by a factor of $\|H\|$, where $\|.\|$ denotes the Frobenius norm. The same holds true about the eigenvalues. In other words, if the perturbation in the graph structure is sufficiently small, the perturbed eigenvalues will not be zero but remain the smallest eigenvalues of $\tilde{L}$ and their corresponding perturbed eigenvectors will point close enough to the original vertices of the hypercube to be separable by a suitable technique, e.g. k-means algorithm[22].

Equipped with the above information we may proceed to the spectral clustering algorithm itself. It was proposed in [23] and we quote a slightly different version from [22].

Input: Similarity matrix $S \in \mathbb{R}^{n\times n}$, number k of clusters to construct.

1. Construct a similarity graph with adjacency matrix W.
2. Compute the normalized Laplacian L_{sym}.
3. Compute the first k eigenvectors $u_1, \dots u_k$ of L_{sym}.
4. Let $U \in \mathbb{R}^{n\times k}$ be the matrix containing the vectors $u_1, \dots u_k$ as columns.
5. Form the matrix $T \in \mathbb{R}^{n\times k}$ from U by normalizing the rows to norm 1, that is set $t_{ij} = u_{ij}/(\sum_k u_{ik}^2)^{1/2}$.
6. For $i = 1, \dots n$, let $y_i \in \mathbb{R}^k$ be the vector of the i-th row of T.
7. Cluster the points $(y_i)_{i=1,\dots n}$ with the k-means algorithm into clusters $C_1, \dots C_k$.

Output: Clusters $A_1, \dots A_k$ with $A_i = \{j | y_j \in C_i\}$.

The algorithm was developed to cluster arbitrary objects, for which we can find a similarity function, therefore the first step is to construct a similarity graph. In our case, we use directly the graph we already have.

[1]Note that while H_A and H_W are random, H_L has to be calculated from the former ones.

K-means clustering algorithm

The goal of K-means clustering algorithm is to divide the observations into K groups, such that the members of the groups are similar. To measure the similarity Euclidian distance is used. Following, more formal, description is adopted from [15].

Suppose there is a function $C(i) = k$, which assigns each observation i to k, one of K clusters. K-means clustering algorithm aims to minimize the within-cluster point scatter defined as

$$W(C) = \frac{1}{2} \sum_{k=1}^{K} \sum_{C(i)=k} \sum_{C(i')=k} \|x_i - x_{i'}\|^2. \tag{65}$$

In other words the algorithm minimize the Euclidean distance between observations inside individual clusters. Equation 65 may be rewritten as

$$W(C) = \sum_{k=1}^{K} \sum_{C(i)=k} \|x_i - \bar{x}_k\|^2, \tag{66}$$

where $\bar{x}_k$ is a mean of observations within cluster k. Now we are looking for such an assignment which would minimize the within-cluster point scatter

$$C^* = \min_C \sum_{k=1}^{K} \sum_{C(i)=k} \|x_i - \bar{x}_k\|^2. \tag{67}$$

Noting that for any set of observations S it holds that

$$\bar{x}_S = \arg\min_m \sum_{i \in S} \|x_i - m\|^2 \tag{68}$$

we may obtain the solution by solving an enlarged problem

$$\min_{C, \{m_k\}_1^K} \sum_{k=1}^{K} \sum_{C(i)=k} \|x_i - m_k\|^2. \tag{69}$$

From the above one may conclude the k-means clustering algorithm[24]:

1. Randomly assign each observation to one of the K clusters.
2. Iterate until there is no change in cluster assignments:
   a) Compute a centroid for each cluster. A centroid of a cluster is a mean of the observations, which are assigned to the cluster.
   b) Change the cluster assignments so that an observation is assigned to the cluster represented by the closest centroid. To evaluate the distance use standard Euclidian metric.

Since the result of this algorithm depend on the initial assignments, it is recommended to repeat it few times and choose the solution with lowest value of $W(C)$.

Estimation of number of clusters

The spectral clustering algorithm presented above takes the number of clusters k as an input argument but usually we have no prior information about how many clusters are present in the graph. We may estimate k from the eigenvalues of L using so called eigengap heuristic.

The idea behind the heuristic is that the eigenvalues $\tilde{\lambda}_i$ of the graph are of form

$$\tilde{\lambda}_i = \lambda_i + h_i, \tag{70}$$

where λ_i is the eigenvalue of the disconnected graph and h_i is the corresponding perturbation. If the perturbation is sufficiently low and the first non-zero eigenvalue λ_{k+1} of the disconnected graph is high enough, the difference $(\tilde{\lambda}_{k+1} - \tilde{\lambda}_k)$ will be relatively big. Therefore to estimate the number of clusters we choose the first k for which $\tilde{\lambda}_{k+1} \gg \tilde{\lambda}_k$[22].

Laplacian comparison

We stated that there are three different Laplacians: L (eq. 56), L_{sym} (eq. 60), and L_{rw} (eq. 61). In this section we discuss, which one of them is the best option for the clustering algorithm.

We start with a few definitions: We denote the sum of weights of edges connecting two sets of vertices $A, B \subset V$ as

$$W(A,B) = \sum_{i \in A, j \in B} w_{ij}. \tag{71}$$

Given the sets of vertices $A_1, \dots, A_k \subset V$, the cut of these sets is

$$\text{cut}(A_1, \dots, A_k) = \frac{1}{2} \sum_{i=1}^{k} W(A_i, \bar{A}_i), \tag{72}$$

where $\bar{A} = \{v \in V \mid v \notin A\}$. A natural requirement for a good clustering algorithm, as we have discussed in the introduction to section 4.1, is to minimize the cut of the produced clusters. Unfortunately, minimizing this particular function may lead to undesired results, such as clusters consisting of one vertex. To avoid this we may enhance the function with a requirement for reasonable large clusters such as with

$$\text{RatioCut}(A_1, \dots, A_k) = \frac{1}{2} \sum_{i=1}^{k} \frac{W(A_i, \bar{A}_i)}{|A_i|} = \sum_{i=1}^{k} \frac{\text{cut}(A_i, \bar{A}_i)}{|A_i|} \tag{73}$$

and

$$\text{NCut}(A_1, \dots, A_k) = \frac{1}{2} \sum_{i=1}^{k} \frac{W(A_i, \bar{A}_i)}{\text{vol}(A_i)} = \sum_{i=1}^{k} \frac{\text{cut}(A_i, \bar{A}_i)}{\text{vol}(A_i)}, \tag{74}$$

where

$$\text{vol}(A) = \sum_{i \in A} d_i \tag{75}$$

is a sum of the degrees of the vertices in set A. It may be shown that minimizing RatioCut is related to spectral clustering with unnormalized Laplacian, while NCut is related to spectral clustering with normalized Laplacian.

Both RatioCut and NCut minimize directly the cut of clusters as we required. Additionally in the spirit of the discussion in the introduction to section 4.1 we may also want to maximize within cluster similarity. Within cluster similarity for cluster A may be denoted as $W(A, A)$. Noting that

$$W(A, A) = W(A, V) - W(A, \bar{A}) = \text{vol}(A) - \text{cut}(A) \tag{76}$$

we can see that within cluster similarity is maximized if $\text{cut}(A)$ is small and $\text{vol}(A)$ is large. That is achieved by minimizing NCut

From the above discussion we see that the normalized Laplacians are more preferable option than the unnormalized Laplacian. Out of the two normalized Laplacians it is recommended to use L_{rw}, since its eigenvectors are cluster indicator vectors, while eigenvectors of L_{sym} are additionally multiplied by $D^{1/2}$ [22].

4.1.2 Modularity clustering

In this section we are going to discuss clustering algorithms based on modularity. If vertices of a graph are divided into groups, modularity measures how likely the vertices inside a particular group to connect to each other and not to the rest of the graph. As such, modularity is a measure of clustering in a similar sense we have presented above. When we study a graph with a known community membership, modularity allows us to reason about structural properties of the graph. When a membership is unknown, we may use modularity to evaluate quality of results of clustering[25].

A formal definition of modularity is as follows: Supposing the vertices of a graph $G = (V, E)$ are divided into k groups $V_1, \dots V_k$, we define matrix $F \in \mathbb{R}^{k \times k}$, such that element e_{ij} of the matrix is a fraction of the edges which connect V_i with V_j. No that we must count each edge only once. The trace of the matrix $\mathrm{Tr}F = \sum_i e_{ii}$ gives the fraction of edges of the graph, which connects members of the same group.

Additionally we define $a_i = \sum_j e_{ij}$, the fraction of edges connected to members of group V_i. If the connections in the graph were random, it would hold true that $e_{ij} = a_i a_j$. Thus the modularity measure is defined as

$$Q = \sum_i (e_{ii} - a_i^2) = \mathrm{Tr}F - \|F^2\|, \tag{77}$$

where $\|.\|$ denotes a sum of the elements of the given matrix[25].

Modularity maximization algorithm by Newman

It turns out that modularity may be used not only to evaluate results of clustering algorithms but we can also perform clustering by directly maximizing modularity itself. To this end, we may employ a simple greedy algorithm proposed in [26]:
Start with N clusters, such that each cluster contains one vertex of the graph.

1. For each pair of clusters compute the change in modularity ΔQ, which would be caused by merging the two clusters (see bellow).
2. Choose the pair with the highest modularity increase (or with the smallest decrease) and merge them. Record the clusters and the modularity.
3. Repeat until there are no clusters to merge.

At the end, go back through the recorded states and select the one with the highest values of modularity.

The change in modularity may be easily computed as $\Delta Q = e_{ij} + e_{ji} - 2a_i a_j$. Modularity may be also changed only by merging clusters which share an edge. So in step 1 of the algorithm we consider only clusters, which do share an edge.

Louvain algorithm

Louvain algorithm is another example of a greedy community detection algorithm based on modularity maximization, which was first proposed in [27]. Modularity may be expresses as

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j), \tag{78}$$

where $k_i = \sum_j A_{ij}$, $m = \frac{1}{2} \sum_{i,j} A_{ij}$, c_i is a cluster containing vertex i, and $\delta(u, v)$ is the Dirac delta function. The basic idea behind the Louvain algorithm is that when we move vertex i which does not belong to any cluster into cluster c, the change in modularity will be

$$\Delta Q = \left[\frac{\sum_{\text{in}} + k_{i,\text{in}}}{2m} - \left(\frac{\sum_{\text{tot}} + k_i}{2m}\right)^2\right] - \left[\frac{\sum_{\text{in}}}{2m} - \left(\frac{\sum_{\text{tot}}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right], \tag{79}$$

where $\sum_{\text{in}}$ is a sum of weights of the edges, which link vertices inside c, $\sum_{\text{tot}}$ is sum of weights of all edges incident to any vertex in c, k_i is a sum of weights of the edges incident to i, $k_{i,\text{in}}$ is a sum of weights of the edges linking i to vertices in c, and m is a sum of weights of all edges in the graph.

Similar expression holds true for removing a vertex from a cluster. It follows that the change in modularity when moving a vertex from cluster c_i to cluster c_j equals to $\Delta Q_i + \Delta Q_j$, where ΔQ_i stands for the change in modularity, when the vertex is removed from c_j and ΔQ_j stands for the change when the vertex is moved into c_j.

The algorithm itself is as follows: Place each vertex of the graph into a separate community.

1. For each vertex i and all its neighbors i evaluate the gain in modularity caused by moving i to the cluster, where j belongs. Move i to the cluster which corresponds with the highest gain in modularity. If no positive change in modularity for i is possible, do not move it. Repeat this procedure until there is no vertex which could be moved.
2. Create a new graph so that the vertices of the new graph correspond to the clusters found in the previous step. Weights of edges in the new graph are given by the sum of the weights of edges, which linked the clusters. Edges linking vertices inside the same cluster are transformed into loops. Then repeat the whole process with the new graph.

The algorithm stops when there are no other changes possible.

5 Detection method

In the following section we present the method to detect Tor communication in network traffic based on an analysis of time patterns in communication and relationships between contacted hosts.

There are several reasons to develop algorithms for Tor detection. While we consider using Tor to be absolutely legitimate for private users, it is rather problematic when ran inside corporate networks. Tor usage may be an indication of a network misuse or even malicious activities, such as data theft. In any case, policies prohibiting Tor are quite common in corporate environment. Therefore network administrators need tools for detecting such activities . Besides, malware has been reported to use Tor to obfuscate its command and control channels [1].

Finally, many malware families are moving their communication to more decentralized schemes[28]. We, therefore, plan to extend the method to detect such decentralized communication channels in the future. Tor is a great starting point for this effort since the Tor protocol is well documented and the Tor project provides a wealth of information about the present and the past state of the Tor network, the number of routers, their properties etc.

5.1 Method overview

Our detection method is based on identification of groups of interest in the communication. The algorithm comprises of the following steps:

Filtering We first identify candidates for Tor routers by examining time patterns in the communication. To achieve this, we train a classifier with high recall and low precision and use it to filter out a substantial number of hosts from the dataset. (See sec. 5.2)

Graph construction The next step is to build a graph which links the remaining hosts on the basis of their mutual communication partners. (See sec. 5.3)

Edge pruning In order to make the structure of the graph more distinct we remove the edges of the graph that we consider insignificant. (See sec.5.3.2)

Clustering Finally, we divide the graph into clusters, which represent different groups of servers. We average the results of the classification from the first step over the clusters and use the averaged values to identify the cluster of Tor routers. (See sec. 5.3.3)

In the rest of the chapter we discuss the algorithm in greater detail starting with a description of our detection objectives and input data in the next section. At the end of the chapter, in section 5.4, we provide evaluation of the algorithm's performance.

5.1.1 Detection objectives

Individual users usually access many different services and their overall traffic patterns are a mixture of characteristics of these services. It is, therefore, possible that Tor communication patterns get lost in the noise created by the other services.

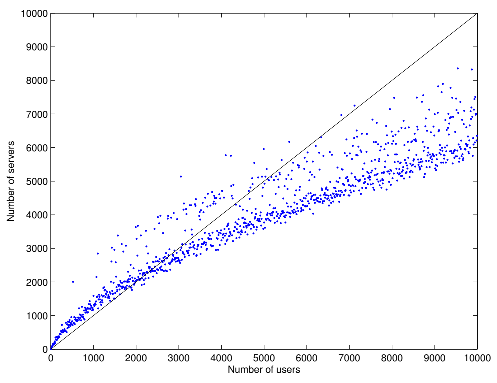


Figure 2 Dependency of the number of servers on the number of users (with highlighted quadrant axis)

```
1417634513319 200 128 0 http://domain.com john_smith 180
123.45.67.890 192.168.0.7
```

Figure 3 An example of a proxy log record

Moreover, users may employ several strategies to change their traffic patterns and conceal the fact they are routing their traffic through Tor. Such cases may be misclassified by a method consider users as the primary target of classification.

We aimed to develop a detection method which is resilient to variations in user traffic. Servers are more likely to be single purpose, e.g. a Tor router; and even if there are users altering their traffic patterns, we expect the majority of users to run standard variants of Tor. In such cases, the portion of normal traffic arriving to Tor routers should be large enough to keep the patterns distinct.

Besides, the number of servers observed in the traffic grows more slowly, than the number of users (as shown in figure 2). When focusing on servers, the amount of information obtained from proxy logs is used to classify fewer entities so statistics computed for them are more reliable.

For these reasons we decided to focus on identification of Tor routers instead of users. Once the routers are identified the knowledge can be easily extended to users who communicate with them.

5.1.2 Input data and the ground truth

The input data for our algorithms were proxy logs of user traffic. Proxy logs contain a wide range of information about connections users make, such as source and destination IP addresses, duration of connections in milliseconds, time stamps when requests took place, downloaded and uploaded bytes, URL, HTTP statuses of requests, user names etc. For us the first three listed above were the most important. An example of a proxy log record is shown in figure 3.

To develop and evaluate the algorithm we used anonymized data from several companies, each with a number of users ranging from tens to thousands. The total duration of the dataset was approximately two weeks.

This amount of data was necessary because Tor is rare in the corporate environment and we would have faced a lack of positive samples if using smaller dataset. It was not possible to process all the data with a regular PC, therefore some parts of the algorithm were performed in a Hadoop cluster.

Tor communicates via encrypted connections and connects directly to the IP addresses of the hosts, which run the routers so it does not use domain names in URL. To reduce the volume of the data we kept only the flows, which were encrypted and headed to raw IP as a preliminary step. This left us with about one percent of the volume of the original data.

After the reduction we ended up with histograms for 1.4 million servers. This is about 100MB of data, which is a four orders of magnitude reduction compared to the original volume. Additionally we discarded all the servers with less than 10 adjacent flows because we did not consider statistics computed from such a small amount of connections reliable. After the last step we had about 230,000 servers to classify. According to our ground truth data, 238 of them were Tor routers.

The source of the ground truth for the evaluation of our experiments was the Tor directory. We considered every server listed in the directory to be a Tor router at the given time. It is actually possible that the Tor router provide other services as well but we do not consider this case probable.

5.2 Classification and filtering

Originally we intended to classify Tor using a well known classification algorithm from the literature, such as Fisher's linear discriminant, SVM, and logistic regression. However none of them provided satisfactory results.

Therefore we propose a two-stage classification algorithm that takes advantage of both classical classification algorithms and well known graph techniques. As the first step we used a readjusted classical classification algorithm that achieves high recall even for the price of low precision and then cleaned the results using graph theory.

We include a short summary of the original effort (the case of SVM classifier) with a discussion why we developed the two-stage algorithm in section 5.2.2.

5.2.1 Features

In section 2.1.2 we have described a circuit rotation employed by Tor. This gives Tor traffic certain temporal properties which we focused on and consequently defined the features for the classification as follows: For each server we make a normalized histogram of duration of flows heading to it. The bins of the histogram are our features. More formally, we choose t_{max}, the maximal time of interest, and N, the number of bins in our histogram. These define a sequence

$$\{t_j\}_0^N, \quad t_j = \delta \cdot j, \tag{80}$$

where $\delta = t_{max}/N$, of boundaries of bins of the histogram. For server S_i we have a set F_i of all connections, which are directed to it. Then x_{ij}, the j^{th} feature of server S_i, is

$$x_{ij} = |\{f \in F_i \ \mid \ \min(t(f), t_{max}) \in (t_{j-1}, t_j]\}| \cdot \frac{1}{|F_i|}, \tag{81}$$

where $t(f)$ denotes the duration of connection f.

We tried to introduce other features: histograms based on downloaded and uploaded bytes, and flow inter-arrival time and their non-linear transformation. We also tried

Table 1 Summary of results of the SVM classifier.

TPs	133	recall	0.56
FNs	105	precision	0.48
FPs	145	f-score	0.52
TNs	235524		

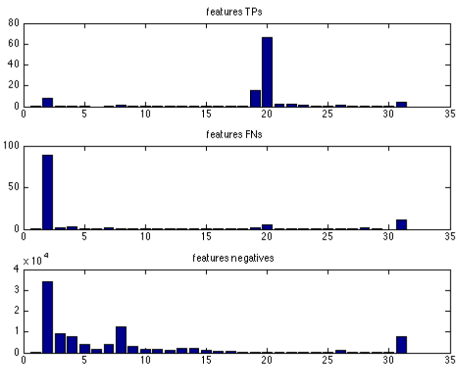


Figure 4 Average duration histograms for the classes of correctly classified Tor (TPs), incorrectly classified Tor (FNs) and other hosts (negatives).

several features based on entropy and some others. We experimented with multidimensional histograms of the features above. We also employed some nonlinear kernels. None of those, however, improved the performance significantly.

5.2.2 Single step classification

In this section we present the results of our initial effort to classify Tor with the SVM classifier (for description of SVM see section 3.2). As we already mentioned, the results were not satisfactory - the recall and precision were 56% and 48% respectively, for details see table 1. The SVM is not part of the final algorithm but we include the discussion of its results to explain why we developed the two-stage classification.

Despite the classifier was not able to find all of the Tor routers, it was able to separate wast majority of host which do not belong to the Tor network. On the figure 4, which plots the weights the classifier assigns to the individual features, we can see that few of the features are assigned significantly higher weights than the others. When we reduce our feature space to the two features which are emphasized most we can visualize the samples, as we do in figure 5. The plot suggests that the Tor routers are indeed clustered together in the feature space. The problem is that the non-tor hosts overlap with the Tor cluster. The SVM places the decision boundary so it minimizes the number of misclassified samples. We might move the decision boundary in a way the classification is suboptimal in the sense of the above metric but which would lead to classification of most of the Tor routers as positive at the cost of increasing the number of false positives. Still we should be able to separate substantial majority of non-Tor hosts and then employ methods described further in section 5.3 to get rid of the huge amount of false positives we introduce to our results.

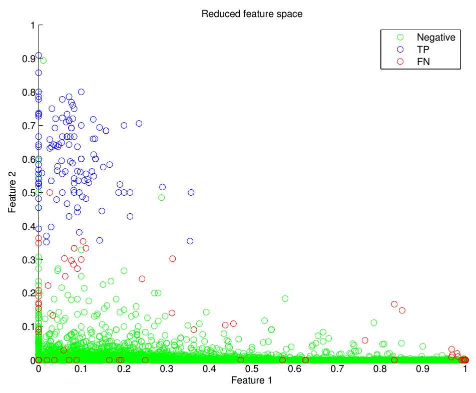


Figure 5 Samples visualized in the reduced feature space. (Negative examples are down sampled.)

Table 2 Summary of the results of the logistic regression classifier with low threshold.

TPs	213	recall	0.89
FNs	25	precision	0.01
FPs	22100	f-score	0.02
TNs	213569		

5.2.3 Filtering

In the previous section we trained SVM to classify the Tor routers and found it insufficient. We concluded that the solution is to move the decision boundary to increase the recall in exchange for precision and cleaning the results later.

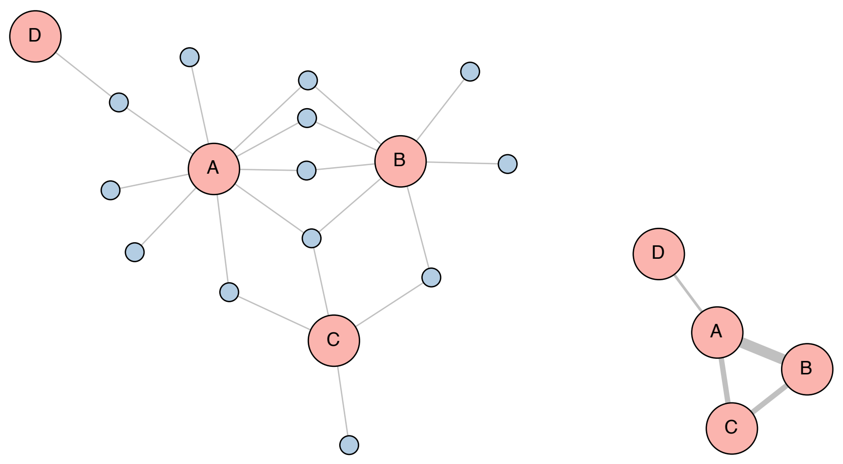
In this section we train the logistic regression classifier (for description of logistic regression classifier see section 3.1) and move the decision boundary.

The reason for the switch to logistic regression is that moving the decision boundary is more straightforward in the case of logistic regression than in the case of SVM[1]. The results we obtained using the logistic regression instead of the SVM for the initial classification problem were slightly worse than those obtained from the SVM but we can afford the change because the precision of the classification is not such a concern any more.

We trained the logistic regression classifier on the same dataset and then lowered the threshold for classifying a sample as positive. The only constrain for our new threshold is the amount of data we are able to process in the next stage of the algorithm. As expected the gain in recall were obtained in expense of ruining the precision of the result. For more details wee the table 2.

It is important to note that from the set of hosts classified as Tor routers only approximately 100 were assigned a high likelihood of being Tor. These are the hosts which we correctly classified as Tor routers in the previous step too. The rest of the hosts has been given likelihood which was close to the chosen threshold. We mention this fact because we will make use of it in section 5.3.3.

[1]We are aware that there exists variants of SVM, which allows to move the decision boundary, e.g. [29].



a) Full **b)** Condensed

Figure 6 Example of communication graph with overlapping sets of users (on the left) and its condensation with edges weighted according to the number of common users (on the right). Servers are marked in red, users in blue. *(Colors from www.ColorBrewer.org by Cynthia A. Brewer, Geography, Pennsylvania State University.)*

5.3 Mutual contacts

As the second step in our classification framework we use graph theoretical methods in order to separate the Tor routers from the rest of the hosts.

The steps are as follows:

- We define a graph, which describe the communication patterns of the hosts in section 5.3.1
- We prune the edges of the graph to make its structure more distinct in section 5.3.2
- We cluster the graph and identify the cluster of the Tor routers in section 5.3.3
- The results are evaluated in section 5.4

5.3.1 Graph definition and creation

The reason to use the graph theory is following: The users do not choose Tor routers with uniform probability. Some of the routers are significantly more popular than the others. Considering the fact the users change the circuits (and corresponding entry routers) often and even open a number of circuits preemptively (see section 2.1.2), and therefore communicate with a high number of Tor routers, we assume there is a high probability for a user to communicate with a number of those highly popular routers.

Also the popular routers, obviously, have a high number of users connecting to them. Combining this assumptions we conclude that it is likely that any given pair of popular Tor routers will share common users. If the assumption is correct and the number of shared users is high enough, it should be possible to link the Tor routers through the users they share. Illustration of such a situation is shown in figure 6a.

Following the intuition above we define the mutual communication graph. The vertices of the graph are the servers, which were assigned sufficient likelihood of being Tor by the filtering step of the algorithm. We connect a pair of servers with an edge if they

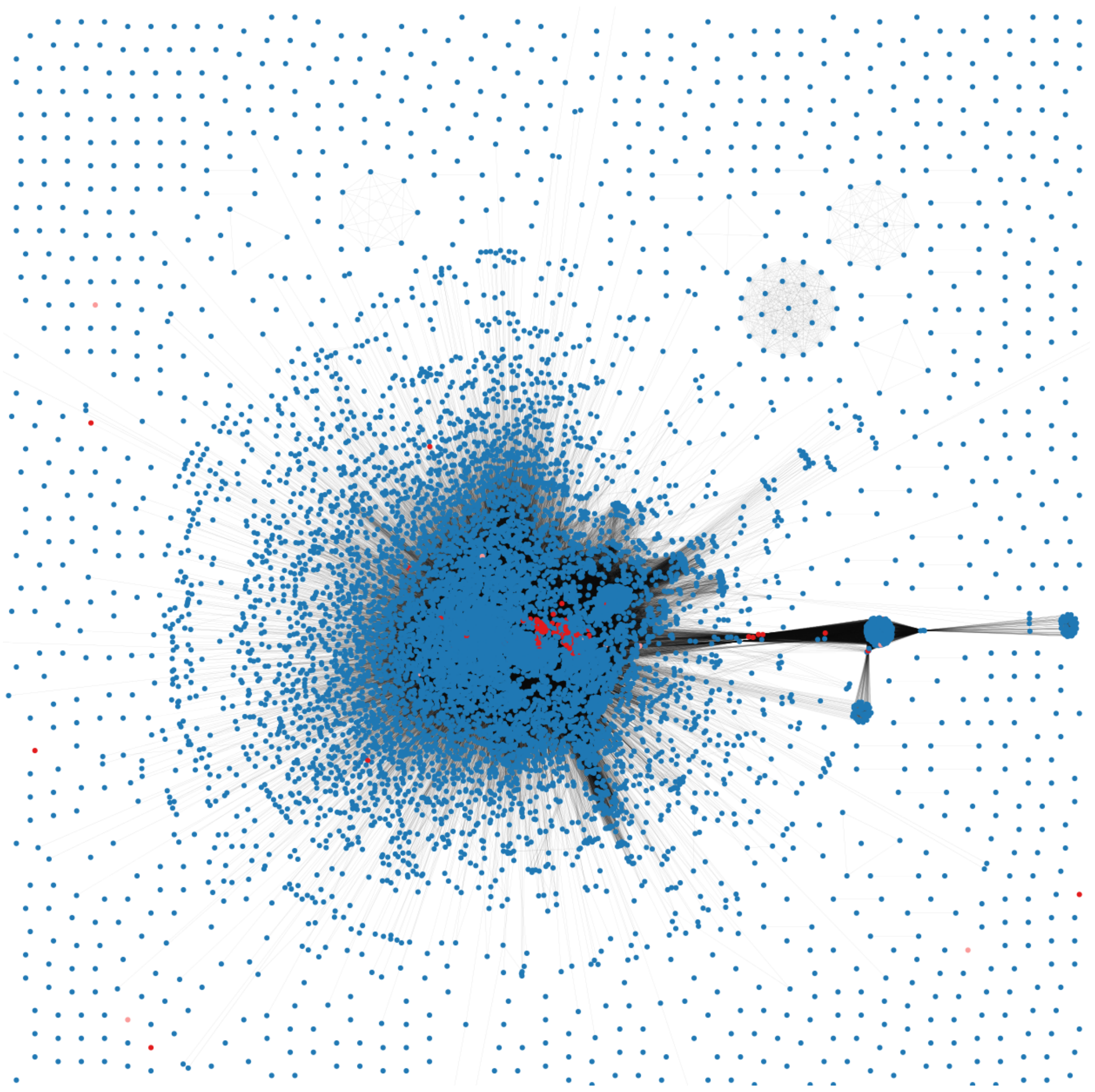


Figure 7 Mutual contacts graph. The Tor nodes are marked in red. *(Colors from www.ColorBrewer.org by Cynthia A. Brewer, Geography, Pennsylvania State University.)*

share a common user and assign a weight to the edge to represent the similarity of the user sets of the respective servers. As a similarity measure we use Jaccard index of the sets[2]. An example of such a graph is in figure 6. The left side of the figure depicts the original communication pattern, where users are linked to the servers they communicate with, and the right side of the figure shows the corresponding mutual communication graph.

More formally, we have S, the set of servers, $p(s)$, the likelihood that a server $s \in S$ is a Tor router, estimated by the classifier in the previous step of the algorithm, and the threshold t. We define the mutual communication graph G as follows:

$$G = (V, E, w) \tag{82}$$

$$V = \{s \in S \mid p(s) \geq t\} \tag{83}$$

[2]Jaccard index of sets A, B is defined as $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$

$$E = \{(s_i, s_j) \in V \times V \ \mid\ w(s_i, s_j) > 0\} \tag{84}$$

$$w(s_i, s_j) = \frac{|U_i \cap U_j|}{|U_i \cup U_j|}, \tag{85}$$

where U_i denotes the set of users of the server s_i.

Visualization of the graph defined above, which was generated from our dataset, is in figure 7. There is a number of isolated nodes and few small, highly interconnected components but the majority of nodes is part of one giant component. Even the giant component is interconnected so tightly there are no obvious clusters to separate. On the other hand, when we mark the Tor nodes by red color, we see that almost all of them are actually gathered on one location in the graph.

5.3.2 Edge pruning

In the previous section we obtained mutual communication graph with tightly connected giant component. To make the clusters in the graph more apparent we would like to remove insignificant edges. Since we already defined edge weight in a way it describes similarity between the servers a natural step is to remove the edges with the lowest weight.

As we raise the threshold for the edge removal, individual nodes and new components are separating from the already existing components. The existing components, especially the giant component, become more structured and we can observe emerging clusters of nodes. When the threshold exceeds a certain value the giant component disintegrates completely.

Threshold value 0.2 and above seems to be sufficient to clear enough edges, which obscure the structure of the graph and to reveal the clusters. Values near 0.5 leaves the graph mostly broken into small pieces.

Figure 8 depicts the graph after the edges with lowest weights were pruned with threshold 0.2. Additionally figure 9 shows the changes in the giant component of the graph according to the pruning threshold.

5.3.3 Clustering

From the visualizations we made in the previous sections we know that the Tor relays stay gathered in the pruned graph. Now we step beyond the visual inspection and try to separate clusters of the graph algorithmically and to identify the clusters which contain the Tor relays.

So far, the graph is divided into isolated nodes and components of various sizes with most of the nodes belonging to the giant component. We discard all the isolated nodes as they do not comply to our model of Tor as an interconnected network. Admittedly doing this we may miss some of the Tor routers, but we believe that these cases will be only the routers of lower importance. The small components already seem to represent relevant groups but we still need to divide the big ones, especially the giant component, into smaller parts.

One possible way to do this would be to raise the pruning threshold until the cluster of Tor routers divides from the graph as we have seen in the experiments. The problem is that we do not know how to choose the threshold and choosing the threshold too high would destroy also the cluster we are looking for. Our approach is to employ one of the graph clustering algorithms described in section 4.1.

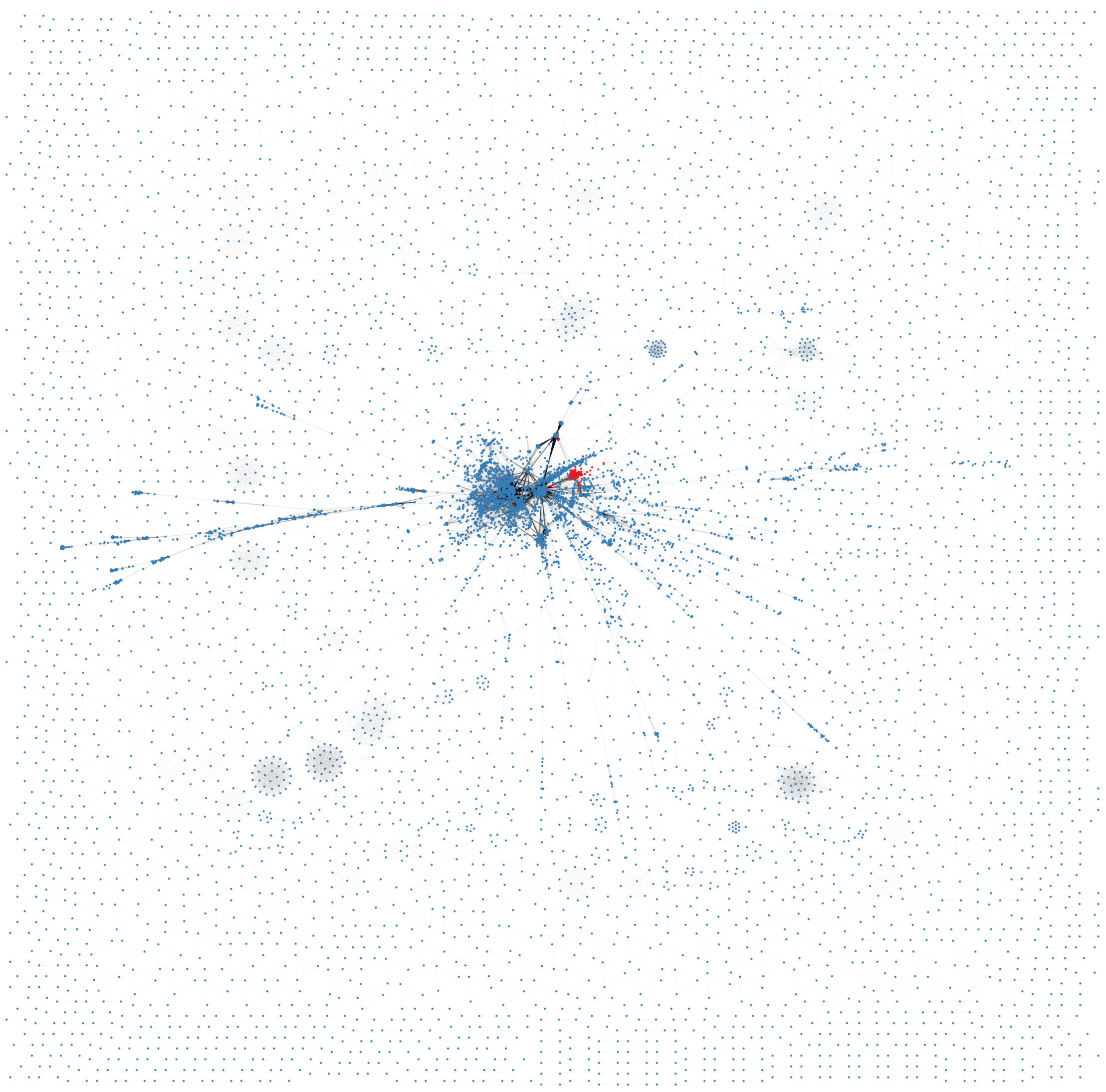
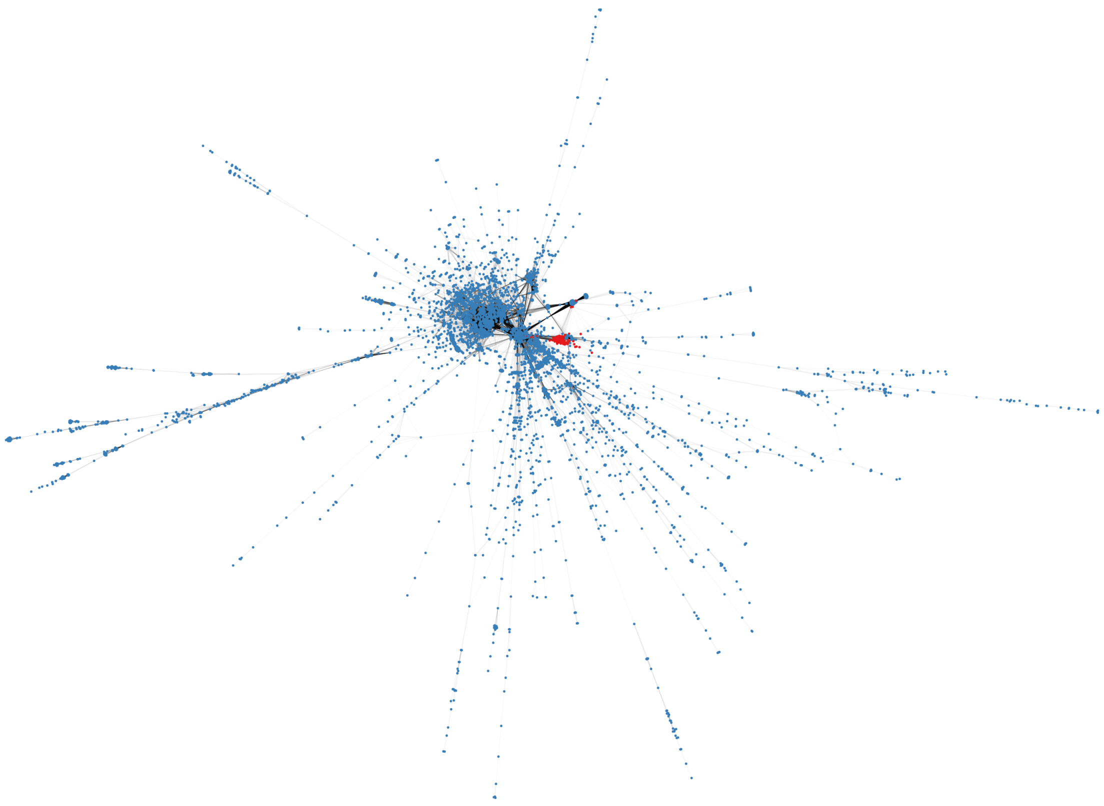


Figure 8 Mutual contacts graph after edge pruning with threshold 0.2. Number of new components detached from the giant component and the rest seems to be significantly better structured. (The Tor nodes are marked in red.) *(Colors from www.ColorBrewer.org by Cynthia A. Brewer, Geography, Pennsylvania State University.)*

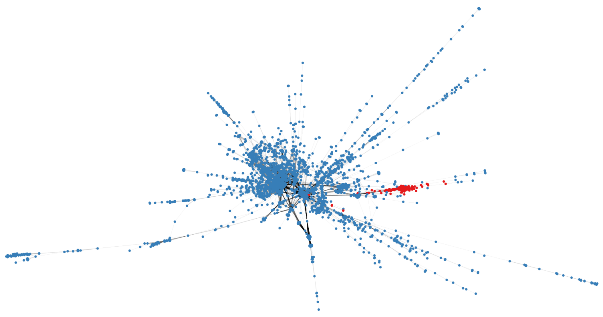
We expected that the clustering algorithms will perform best when using the weight of the edges, after all the weights reflect the similarity of the servers they connect. Surprisingly, this approach did not work well and we obtained much better results when we changed the weights of the edges, which remained in the graph after the pruning, to 1.

We experimented with spectral clustering and Louvain clustering algorithms. Both algorithms performed well on our graph in the sense that they were both able to catch its structure and divide it into meaningful clusters. However they produced significantly different number of clusters.

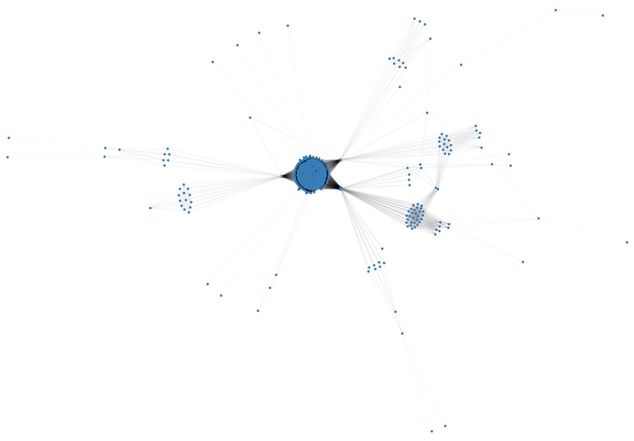
While the heuristics for the estimation of the number of clusters we used with spectral clustering (see sec. 4.1.1) estimated there were 8 clusters in the giant component, the number of cluster yielded by the Louvain clustering was about two orders of magnitude higher. Since we prefer to have smaller, well defined clusters we adopted Louvain clustering into our algorithm. The results of spectral clustering are shown in figure 10.



a) Threshold 0.2



b) Threshold 0.3



c) Threshold 0.5

Figure 9 Comparison of the giant component of the graph with different edge pruning threshold. Note that the component containing Tor nodes (marked in red) detached from the giant component for the pruning threshold 0.5. (All three figures are at the same scale.)

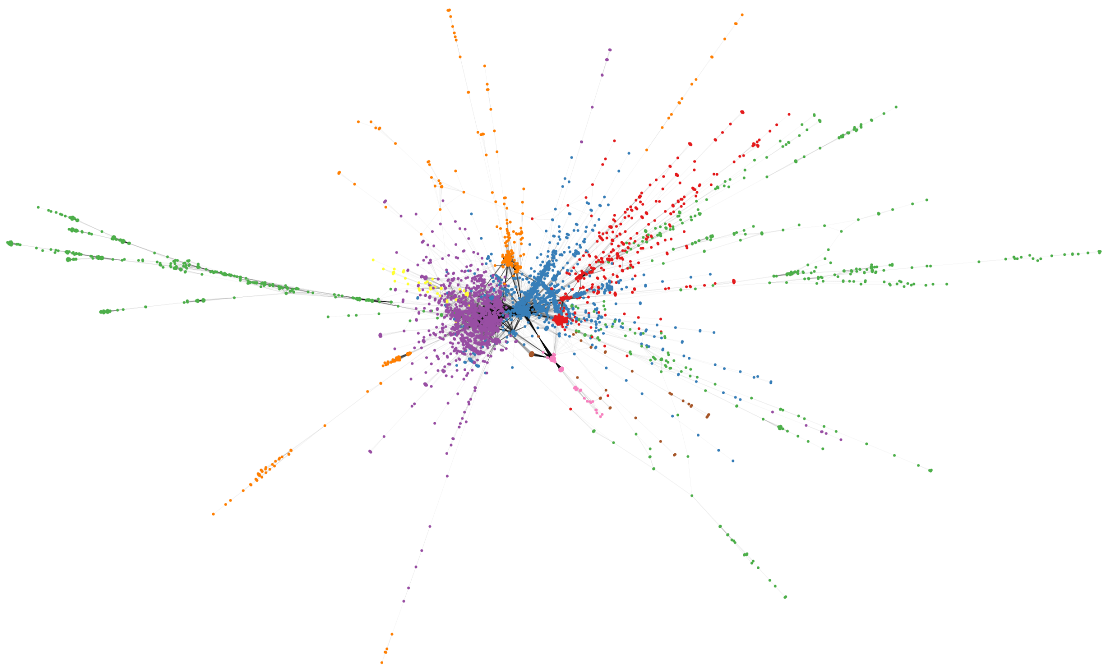


Figure 10 Visualization of the giant component divided into 9 clusters. The majority of Tor routers is located in the red cluster but the granularity of the clustering is not sufficient to separate them completely from the other nodes. *(Colors from www.ColorBrewer.org by Cynthia A. Brewer, Geography, Pennsylvania State University.)*

We do not include figure with the results of Louvain clustering due to the high number of clusters it found.

5.3.4 Tor cluster identification

After dividing the graph into small components and clusters, we proceed to the identification of the cluster of Tor routers. For each group of nodes (cluster or component) we compute an average of the likelihood which was assigned by the logistic regression classifier to the nodes it contains. Then we choose the group with the highest average value to represent the cluster of Tor routers, i.e., we assign positive label to all nodes in the group and negative label to all nodes in the other groups.

More formally: We have the mutual contacts graph $G = (V, E, w)$, the likelihood $p(s)$ of being Tor assigned to a host $s \in V$ by the classifier in the first step of the algorithm, and the cluster assignment function $c(s)$, which assigns a cluster number to each node according to the results of the clustering described above. Set

$$C_i = \{s \in V \mid c(s) = i\} \tag{86}$$

is the set of nodes in cluster i. Then we compute $p(C_i)$, the average likelihood that the cluster i is a cluster of Tor nodes, as

$$p(C_i) = \frac{1}{|C_i|} \sum_{s \in C_i} p(s). \tag{87}$$

We identify the cluster of Tor routers C_{tor} by choosing the cluster with highest average likelihood

$$C_{\text{tor}} = \arg\max_{C_i} \; p(C_i) \tag{88}$$

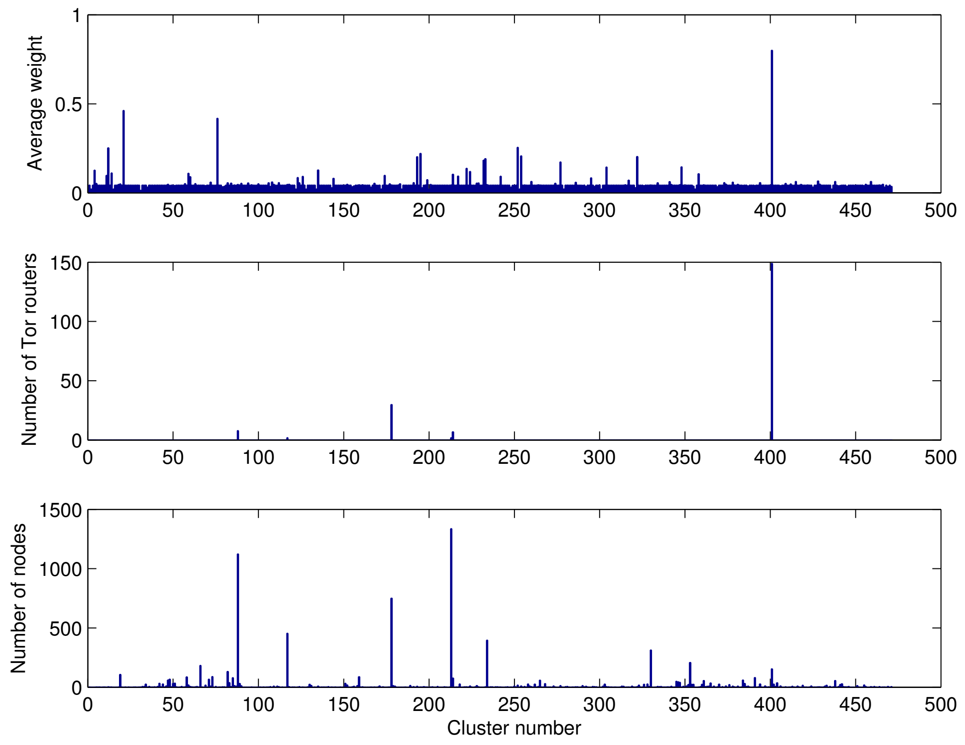


Figure 11 Details of clustering results: the average likelihoods (denoted as weights in the graph), number of Tor routers and number of all nodes in the clusters.

Table 3 Summary of the parameters we used in the algorithm.

N	30
$t_{\max}$	5min
$V_{\max}$	20,000
$p_{\min}$	0.042
$w_{\min}$	0.2

and we assign labels to individual nodes

$$l_j = \begin{cases} 1 & s_j \in C_{\text{tor}} \\ 0 & \text{otherwise}, \end{cases}$$

where $l_j = 1$ means that we consider the node s_j to be a Tor router.

5.4 Evaluation

For the logistic regression we set the number of bins, N, to 30 and the time cutoff, $t_{\max}$, to 5 minutes. We choose the maximal number of nodes in the mutual contacts graph, $V_{\max}$, to be 20,000, consequently the classification threshold of a logistic regression classifier, $p_{\max}$, is set to 0.042. From the experiments with the graph pruning we concluded that the threshold weight of the edge, $w_{\min}$, equal to 0.2 is sufficient to reveal the clusters without loosing too much information about the mutual contacts. The parameters are summarized in table 3.

Table 4 Summary of the results of our detection method.

TPs	149	recall	0.63
FNs	89	precision	0.96
FPs	6	f-score	0.75
TNs	235663		

Table 5 Results of our detection method compared to results we obtained using SVM classifier.

	SVM	our method
recall	0.56	0.63
precision	0.48	0.96
f-score	0.52	0.75

Table 6 Clusters sorted by their average weight with number of Tor routers and a total number of nodes they contain.

	Weight	Tor nodes	Total nodes
1	0.80	149	155
2	0.46	0	2
3	0.42	0	2
		...	
22	0.10	7	78
		...	
77	0.05	8	1125
		...	
93	0.04	30	752

With these parameters the cluster identified by the algorithm as the Tor cluster contains 149 out of the 238 Tor relays, which are in the dataset. It also contains 6 non-tor nodes. The detailed summary of the results is in table 4. Table 5 compares the results of our method with the results we obtained using SVM classifier. To conclude it, we have succeeded in increasing the precision of the result while keeping the recall unchanged.

The figure 11 shows weights, number of Tor routers and total number of nodes in all clusters the algorithm found. Precise numbers for some selected clusters are in table 6. As you can see in the figure, the margin in average likelihood of being the Tor cluster between the first and the second cluster is quite large. Also there is a significant number of clusters without any Tor, which precede the Tor cluster with second largest weight.

Significant number of Tor routers is not part of the main Tor cluster. Figure 11 that there are two or three more clusters which contain non-negligible amount of Tor nodes. Unfortunately, those clusters also contain hundreds of other nodes so including them to the positively classified nodes would ruin the results completely.

One possible direction for the future improvement of the algorithm is to investigate why those nodes ended up in different clusters and to develop a way of edge pruning which would stress the connection to the Tor routers and mitigate the relationships with the other nodes.

Another possible explanation of the misclassification is that the nodes which are labeled as Tor routers in our ground truth (and they indeed are Tor routers since they are listed in the Tor directory) run also other services and users access this services instead of Tor. If this is the case, the improvement of our methodology for label creation would improve the results.

6 Related work

Since our proposed method is based on community detection in graphs, we will focus on methods using graph theory.

The method proposed in [30] shares the same basic idea with our method, namely that related hosts are likely to share mutual contacts. The authors define a mutual contacts graph similar to our but its edge weights are based on the total number of shared contacts, not the similarity of the whole contact sets as in our graph. The method requires at least one host to be identified in advance and this knowledge is then spread to other hosts, which are strongly linked to the original one. To this end, the authors propose so called *Dye pumping algorithm*, which is an analogy to a distribution of a dye indicating the relationship between the host from the seed host along the edges of the graph and as such it is a kind of a diffusion process.

Paper [31] propose a method for detecting P2P networks. The authors suppose that the members of P2P networks keep a listening port open for incoming connections and that the connections to it are either long lasting or reoccurring. Following this idea their algorithm finds persistent hosts and builds a bipartite graph from their traffic. The partitions of the graph constitute of (IP, port, protocol) triples representing the local or the remote hosts. An edge is placed between the hosts if communication occurs between them. The remote host partition is further split into suspicious and confirmed group. A new host is initially placed into the suspicious group and to be considered confirmed it must share a significant amount of contacts with another confirmed host or the seed. The resulting graph is divided into number of components and hosts in each component are considered to be part of one P2P network.

A concept of *Traffic dispersion graphs* is used in [32] to classify backbone traffic. The flows from the traffic are grouped based on their features, including first bytes of the flow. The authors claim each group should contain flows generated by one application. From each group a graph is built, where two hosts are linked if a flow between them is observed. The resulting graphs are then classified on the basis of various graph related features.

Paper [33] propose a method for detection of P2P bots in a waiting stage. The authors argue that in order to maintain the botnet overlay network the bots have to communicate frequently. Such signaling should have relatively low volume. In order to observe such a behavior the authors introduce concept of *Superflows*, which aggregate the duration and volume of the traffic between hosts. From Superflows a graph is built where the hosts are linked if they share a Superflow with relatively low volume. This graph is clustered and out of the discovered communities those with high fraction of long lasting connections are selected. Finally those communities are filtered so that only the hosts with long lasting Superflows are left. Those are reported to be members of P2P networks.

The method from [34] leverage mixing properties of random walks in a graph created from backbone traces to reveal possible P2P traffic. Sybil Infer algorithm is then employed to make its results more precise.

Paper [35] uses a graph theoretic approach to define a few of the heuristics it proposes to classify network traffic and it combine them with other approaches including packet inspection. It also uses concept of neighborhood to extend classification of one host to others with similar role in the network.

Similar to [32] our method perform hosts classification as an initial step but unlike the paper we use the classification to filtering the vast majority of hosts out of the process. Same as [31, 33] our method leverage timing patterns to identify the hosts of interest but our method learns the pattern form training set in contrast to fixed threshold used in the papers. Also, our method does not need an initial seed as the method from [30] and we do not need visibility to backbone traffic as [32, 34]. We believe that when weighting or including an edge according to an absolute number of mutual contacts as in [30, 32], hosts with high number of contacts may be falsely linked by coincidence. We try to solve this issue by comparing the whole sets of contacts of the respective hosts. Apart from community detection, we do not evaluate the properties of the constructed graph as in [32], even though it may be an interesting topic for the future research.

Even though some of the methods described above have a different purpose from the method of ours we find it reasonable to compare our method with them. Methods designed directly for Tor usually do not consider detection of the communication itself. It is not even a goal of Tor to hide the fact the communication is carried out (See section 2.1). Traffic confirmation, i.e. showing that two particular parties communicated with each other, or other means of deanonymization of users are the usual subjects of papers analyzing the Tor communication.

Clearly, the most straightforward way to detect Tor communication is to compare the endpoints of the communication with the published list of Tor routers (See section 2.1). Although there exist unpublished Tor routers[4], we believe that using them would not be common and if used some other means might be used to detect them since such a communication would be more stable than the usual. The reason we decided to design this considerably more complicated way of detecting Tor is that we would like to generalize the method to other hidden communication channels, such as those used by malware with decentralized command and control structures.

7 Conclusion

In this text we proposed a method for Tor detection which combines a common machine learning approach with graph clustering.

SVM shows to be insufficient to identify Tor and both the precision and recall of the classifier were low. From the analysis of Tor we concluded that its relays should be linkable due to the high number of communication partners they share with each other. For that reason we proposed a two stage approach. First, we use a logistic regression classifier to reveal possible Tor hosts and discard majority of the others. During this step we can tolerate low precision as long as the recall is acceptable. In the second step, we refine the results by constructing a mutual contacts graph. We isolate clusters of this graph and identify the cluster, which probably contains Tor hosts. All the hosts in the cluster are considered Tor relays.

We evaluated the method using real data from a corporate network with promising results. While the recall was only slightly better than the one of the original approach, the precision improved significantly. Majority of hosts within the identified cluster indeed belonged to Tor. On the other hand, there were a number of Tor relays outside the cluster. There were two or three more clusters, which contained a non-negligible amount of the Tor hosts. Unfortunately, those clusters also contained hundreds of other hosts so including them in the positively classified hosts would have ruined the results. Apart from the method itself, we provided an overview of a number of anonymity networks and discussed the operation of Tor in detail. We also covered some selected theory relevant to the classification and the graph clustering.

The results of our detection method were encouraging but the work may be extended in many directions. First of all, the approach is partially Tor specific because it uses Tor's distinctive timing patterns in the pre-filtering step. As we have discussed in chapter 2, many anonymity networks have properties similar to Tor, therefore our intuition is that it should be possible to generalize the method to detect other networks as well. Another interesting direction would be to verify whether the method is applicable to malware signaling structures.

Furthermore, introduction of graph theory into the problem opens wide variety of options. Most evident is the question: What are the other clusters and components in the graph? Is there any useful information, which can be inferred from them? We believe that the answer is yes. Second, despite the edge weighting function proved to be useful, we think it may be improved to stress the relationship between the hosts we are interested in and suppress those relationships, which cause unwanted artifacts in the graph.

Bibliography

[1] Dmitry Tarakanov. *The Inevitable Move - 64-bit ZeuS Enhanced With Tor.* 2013. URL: `http://securelist.com/blog/events/58184/the-inevitable-move-64-bit-zeus-enhanced-with-tor/` (visited on 12/03/2014).

[2] Roger Dingledine, Nick Mathewson, and Paul Syverson. *Tor: The second-generation onion router.* Tech. rep. DTIC Document, 2004.

[3] Steven Murdoch and Nick Mathewson. *Top changes in Tor since the 2004 design paper (Part 1).* 2012. URL: `https://blog.torproject.org/blog/top-changes-tor-2004-design-paper-part-1` (visited on 12/03/2014).

[4] Steven Murdoch and Nick Mathewson. *Top changes in Tor since the 2004 design paper (Part 2).* 2012. URL: `https://blog.torproject.org/blog/top-changes-tor-2004-design-paper-part-2` (visited on 12/03/2014).

[5] George Danezis, Roger Dingledine, and Nick Mathewson. “Mixminion: Design of a type III anonymous remailer protocol”. In: *Security and Privacy, 2003. Proceedings. 2003 Symposium on.* IEEE. 2003, pp. 2–15.

[6] Pere Manils et al. “Compromising tor anonymity exploiting p2p information leakage”. In: *arXiv preprint arXiv:1004.1461* (2010).

[7] Dennis Fisher. *Researcher Finds Tor Exit Node Adding Malware to Binaries.* 2014. URL: `http://threatpost.com/researcher-finds-tor-exit-node-adding-malware-to-binaries/109008` (visited on 01/04/2015).

[8] Roger Dingledine and Nick Mathewson. “Anonymity Loves Company: Usability and the Network Effect.” In: *WEIS.* 2006.

[9] Michael K Reiter and Aviel D Rubin. “Crowds: Anonymity for web transactions”. In: *ACM Transactions on Information and System Security (TISSEC)* 1.1 (1998), pp. 66–92.

[10] Brian Neil Levine and Clay Shields. “Hordes: a multicast based protocol for anonymity”. In: *Journal of Computer Security* 10.3 (2002), pp. 213–240.

[11] Bassam Zantout and Ramzi Haraty. “I2P data communication system”. In: *ICN 2011, The Tenth International Conference on Networks.* 2011, pp. 401–409.

[12] Ian Clarke et al. “Freenet: A distributed anonymous information storage and retrieval system”. In: *Designing Privacy Enhancing Technologies.* Springer. 2001, pp. 46–66.

[13] Jinsong Han and Yunhao Liu. “Rumor riding: Anonymizing unstructured peer-to-peer systems”. In: *Network Protocols, 2006. ICNP’06. Proceedings of the 2006 14th IEEE International Conference on.* IEEE. 2006, pp. 22–31.

[14] Li Zhuang et al. “Cashmere: Resilient anonymous routing”. In: *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2.* USENIX Association. 2005, pp. 301–314.

[15] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning.* Vol. 1. Springer Series in Statistics New York, 2001.

[16] Andrew Ng. *CS229 Lecture notes*. 2012. URL: `http://cs229.stanford.edu/notes/cs229-notes1.pdf` (visited on 01/04/2015).

[17] Christopher M Bishop et al. *Pattern recognition and machine learning*. Vol. 1. springer New York, 2006.

[18] Reinhard Diestel. *Graph Theory {Graduate Texts in Mathematics; 173}*. Springer-Verlag Berlin and Heidelberg GmbH & Company KG, 2000.

[19] Mark Newman. *Networks: an introduction*. Oxford University Press, 2010.

[20] Santo Fortunato. "Community detection in graphs". In: *Physics Reports* 486.3 (2010), pp. 75–174.

[21] Satu Elisa Schaeffer. "Graph clustering". In: *Computer Science Review* 1.1 (2007), pp. 27–64.

[22] Ulrike Von Luxburg. "A tutorial on spectral clustering". In: *Statistics and computing* 17.4 (2007), pp. 395–416.

[23] Andrew Y Ng, Michael I Jordan, Yair Weiss, et al. "On spectral clustering: Analysis and an algorithm". In: *Advances in neural information processing systems* 2 (2002), pp. 849–856.

[24] Gareth James et al. *An introduction to statistical learning*. Springer, 2013.

[25] Mark EJ Newman and Michelle Girvan. "Finding and evaluating community structure in networks". In: *Physical review E* 69.2 (2004), p. 026113.

[26] Mark EJ Newman. "Fast algorithm for detecting community structure in networks". In: *Physical review E* 69.6 (2004), p. 066133.

[27] Vincent D Blondel et al. "Fast unfolding of communities in large networks". In: *Journal of Statistical Mechanics: Theory and Experiment* 2008.10 (2008), P10008.

[28] David Dittrich and Sven Dietrich. "New directions in peer-to-peer malware". In: *Sarnoff Symposium, 2008 IEEE*. IEEE. 2008, pp. 1–5.

[29] J Platt. *Probabilistic output for support vector machines and comparisons to regularize likelihood methods. Advanced in Large Margin Classifiers*. 2000.

[30] Baris Coskun, Sven Dietrich, and Nasir Memon. "Friends of an enemy: identifying local members of peer-to-peer botnets using mutual contacts". In: *Proceedings of the 26th Annual Computer Security Applications Conference*. ACM. 2010, pp. 131–140.

[31] Jan Jusko and Martin Rehak. "Revealing cooperating hosts by connection graph analysis". In: *Security and Privacy in Communication Networks*. Springer, 2013, pp. 241–255.

[32] Marios Iliofotou et al. "Graption: A graph-based P2P traffic classification framework for the internet backbone". In: *Computer Networks* 55.8 (2011), pp. 1909–1920.

[33] Huy Hang et al. "Entelecheia: Detecting p2p botnets in their waiting stage". In: *IFIP Networking Conference, 2013*. IEEE. 2013, pp. 1–9.

[34] Shishir Nagaraja et al. "BotGrep: Finding P2P Bots with Structured Graph mbox-Analysis." In: *USENIX Security Symposium*. 2010, pp. 95–110.

[35] Thomas Karagiannis, Konstantina Papagiannaki, and Michalis Faloutsos. "BLINC: multilevel traffic classification in the dark". In: *ACM SIGCOMM Computer Communication Review*. Vol. 35. 4. ACM. 2005, pp. 229–240.