Freenet Darknet Mapping
Master Thesis

K.C.N. Halvemaan
University of Amsterdam
kees.halvemaan@os3.nl

July 24, 2017

Abstract

Freenet is a censorship resilient *peer-to-peer* (P2P) file sharing network. Due to the sensitive nature of the content users might attempt to share it is important that an adversary cannot find out who participates in the network or what was shared. A detection method for the IP addresses of Freenet darknet nodes is introduced based on the characteristic IP packet lengths of the application. This research confirms the feasibility of detecting Freenet nodes and mapping a Freenet darknet. In addition, some considerations are given as future work to make it more difficult to do traffic analysis on Freenet.

1 Introduction

Freenet is an anonymous P2P distributed information storage and retrieval system [Clarke et al., 2001]. Content is uploaded to a node which then gets mirrored over the network. The censorship resilient feature of Freenet is found in the fact that the original uploading node can disconnect from the network, but users will still be able to access content the node had uploaded as it is mirrored on various other nodes. This decentralised storage makes it difficult for content to be removed and prevents a single point of failure from crippling the network.

A user will send a request for a file to a node which will then forward the query through the network until the file has been found or the maximum depth has been reached. If a file is found, it is sent to the requesting node, and each intermediate node in the path also saves the file. Thus, the nodes do not know if the request originated from the previous node, or some other node in the network. When no file is found a message will be returned that the query has failed. Various applications have been created for Freenet: websites (so-called Freesites), forums, instant messaging, e-mail and microblogs.

A node can connect to peers through both the opennet and the darknet systems. The former involves a centralised approach where node identities are announced through so-called seed nodes. In the latter decentralised approach a node will only connect to a peer if both sides trust the connection, also known as a *friend-to-friend* (F2F) network. The darknet should have a higher level of privacy as not all nodes are publicly listed, and has been the preferred method since Freenet version 0.7 [Biddle et al., 2002].

The nature of opennet leaves the network vulnerable to mapping techniques [Roos et al., 2014]. This paper will look into the possibility of mapping the nodes participating in a darknet. As with any technology or research, applications can vary from benign to malignant. In the case a mapping could be made on darknet it would indicate a weakness of Freenet which can be exploited. This could be used by parties to discover the *Internet Protocol* (IP) addresses of nodes and their operators, which would violate the privacy of the users. A more benign application would be to improve the security of the Freenet environment by fixing any possible problems allowing the mapping to be made. The latter is important to secure the privacy of the users participating in the network.

A background is given in Section 3, split into two parts: an explanation on the workings of Freenet in Section 3.1, and Section 3.2 gives an overview of the related work. In Section 4 the experimental setup is explained, as well as the experiments which were done in order to answer the research question, of which the results will be shown in Section 5. A discussion is given in Section 6 and a conclusion in Section 7. Section 8 contains some ideas for future work.

2 Research question

The following research question has been formulated.

- Are Freenet opennet mapping techniques applicable to a Freenet darknet?
- Is it possible to discover the IP addresses of nodes participating in a Freenet darknet?

3 Background

An overview of the workings of Freenet is given in Section 3.1 as well as some of the terminology used in this paper. Related work on Freenet mapping is discussed in Section 3.2.

3.1 Freenet

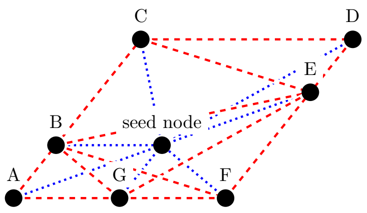
Freenet was first proposed in a white paper by Clarke [1999], and further extended by Clarke et al. [2001]. An improved version was presented in Biddle et al. [2002], fixing a number of security related issues, increasing routing performance, and pushing darknet as the more secure alternative to opennet. The aforementioned papers have been used as a basis for the description of Freenet in this section, as well as the Freenet wiki [Freenet Project, 2017e] and the Freenet website [Freenet Project, 2017d].

Freenet is an overlay network with the Internet as the underlay network, see Figure 2 for a graphical representation. Freenet was designed to be transport protocol agnostic: the original *Freenet REference Daemon* (FRED) used the *Transmission Control Protocol* (TCP), but since version 0.7 it has been using the *User Datagram Protocol* (UDP). The operator of a Freenet node is often its only user. It would create significant security risks if one uses a remote node operated by a third party. With privacy and security in mind the Freenet project discourages this behaviour and encourages running a local node in order to connect to the network. Each node has a unique identity with which it identifies itself to its neighbours. It is exchanged with other nodes in order to peer and set up a secure channel. Part of the identity are the UDP port used for connecting and the public key of the node. Nodes send messages to each other containing queries, data, or in the case of probes special functionality as part of the *Freenet Protocol* (FNP). A message has a *Unique Identifier* (UID) which allows nodes in the network to apply loop-prevention by checking if they have seen a message with a specific UID before. Each node has a location l, with l being a number in the range $[0, 1)$. The location is a circular relation, meaning the distance between $l = 0.1$ and $l = 0.9$ is 0.2. When a node joins the network, its location[1] is set to a random value. Note that the location of a node is unrelated to the location of its neighbours as can be seen in Figure 3 which depicts the same network as Figure 2 but with the nodes positioned on a circle based on their location. When considering the network as a *Distributed Hash Table* (DHT), a Freenet node specialises in storing files that have a key close to its location. The location is used in the routing of queries through the network, as will be explained later in this section.

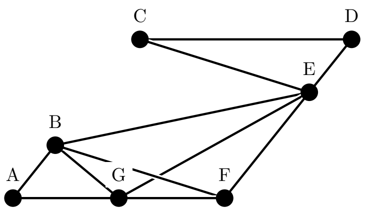
When a user initialises his node a public-private key pair is created, these are then used to sign and verify published files. A number of different keys can be created for a file. The *Content Hash Key* (CHK) which is the hash of the file, is used for integrity checking. The *Keyword Signing Key* (KSK) is a hash created from a human readable string which allows for keyword searches. For Freesites that are updated frequently a *Signed Subspace Key* (SSK) can be used which contains the signature of the author allowing users to validate that an updated version was indeed made by the same author. The usage of SSKs has been superseded by the *Updateable Subspace Key*s (USKs), which are wrappers for the former hiding the search for the most up-to-date version of a Freesite. The public keys can be obtained out-of-band or from Freenet sites that provide indexed lists of said keys.

The network topology in Freenet is created in different ways for opennet and darknet, as can be seen in Figure 1. In opennet all nodes have a special connec-

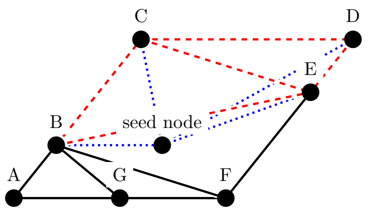
[1]From here on when "location" is used in this paper this property of the node is meant.



(a) Opennet.



(b) Darknet.



(c) Hybrid.

Figure 1: The three possible topologies within Freenet. Solid lines indicate darknet connections, dotted lines are connections to the seed node and dashed lines are connections assigned by a seed node.

tion to a seed node, which helps them connect to a neighbour with a relevant location by forwarding join requests based on the location of the node. Figure 1a shows what a Freenet opennet looks like. Neighbours in opennet can also be removed if their performance is insufficient, this should structure the network in such a way to improve routing. The alternative to opennet is darknet, in which each node has a set of trusted neighbours with which it has made an agreement to peer, an example of this can be seen in Figure 1b. The neighbours have to be added manually by the user. Darknet nodes swap their location with other nodes in order to uniformly spread the key space covered by the network. A hybrid form is also possible, where a darknet is connected to the opennet through a peer which participates in both. Figure 1c displays this hybrid topology. In this case darknet nodes A and G can access information on the opennet node C (and vice versa) through node B which participates in both opennet and the darknet. This way the opennet can act as a connecting component for multiple darknets.

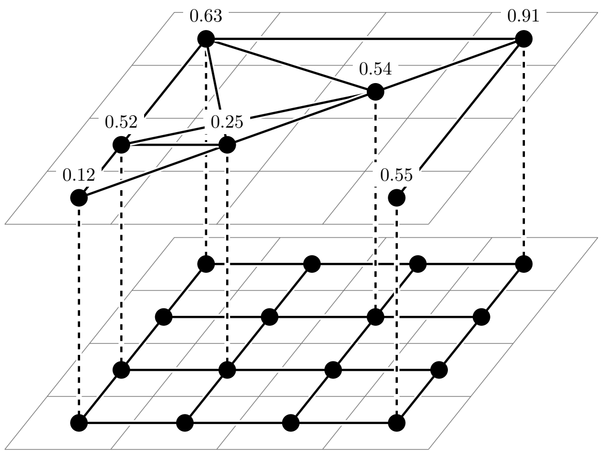


Figure 2: Freenet depicted as an overlay network, the label of the nodes indicate their location.

The routing in Freenet is based on the small-world model in which most individuals "are linked by short chains of acquaintances" [Kleinberg, 2000b]. Since a darknet is a subgraph of a social relationship graph it should concur with this model (users need to know one another out-of-band before they can connect their nodes). Efficient routing can still be done by individual nodes acting strictly on local information [Kleinberg, 2000a]. The retrieval and insertion of files is based on a depth-first search. When a node is able to fulfil a request it will do so, otherwise it will forward the message to the neighbour whose location is closest to the requested key. If a node is unable to forward the request, a response will be sent, allowing the previous node to requests its next neighbour. Before a user can insert a file into the network, it will first send a request to its node to see if the hash of that file is already present in the network. If a negative response is returned, the file is inserted along the same path, being cached by the nodes along the way. Files are split up into blocks of 32 KiB each, allowing for multiple nodes to store different parts of a file. When the storage capacity of a node is reached, random files will be deleted to make room for new ones.

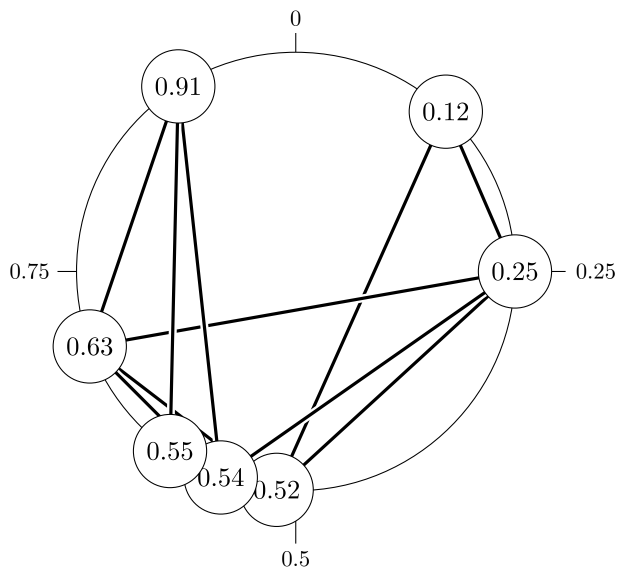


Figure 3: A Freenet topology with the nodes placed in a circle based on their location.

3.2 Related work

Few studies have been solely concentrated on the mapping of Freenet, with it often being a side experiment instead of the main subject. An overview of related work is given in this section, in chronological order.

An opennet measurement was done by Cramer et al. [2004], using a monitoring node they found 19,235 distinct IP addresses during a period of 18 days. In addition, the length of the TCP sessions[2] were measured which had a mean of 34.66 seconds. Of all sessions 99.7% lasted less than 30 minutes and only 0.11% stayed available for more than one hour.

Evans et al. [2007] describes an attack based on the periodical swapping of the location of peers in a darknet. Since the location of a node is separate from the connections to its peers, and swapping occurs naturally as part of the FNP, such an attack is difficult to detect by other nodes. Natural join-leave churn also provides a similar effect as the attack does. This attack has shown major imperfections in Freenet[3], but since the locations and connections are separate it may be difficult to be used for mapping purposes.

Blocking of various P2P applications using Linux iptables was discussed by Othman and Kermanian [2008]. Freenet was blocked by dropping all traffic for specific TCP ports which were used by the FRED. This method cannot be applied to later versions of Freenet since random UDP ports are chosen for the P2P traffic. A similar tactic is used in the commercial *Intrusion Detection System* (IDS) of Solarwinds [2017] where the TCP ports used by the FRED FProxy and the external *Freenet Client Protocol* (FCP) controller are used to flag traffic as malicious. Usage of these TCP ports does not imply that a Freenet node is actually running since these ports are not used for the P2P traffic. In addition, one could change the default ports used by P2P programs to evade this type of rule based blocking.

Vasserman et al. [2009] introduce the term of an *membership-concealing overlay network* (MCON), which concurs with the censorship resilient design principles of Freenet. They discuss resistance to member identification attacks where a MCON outsider tries to retrieve the identities of the members participating in the MCON. Two methods are explained: harvesting, where an adversary tries to find the identity of any number of nodes, and targeting where the attacker tries to match a known identity to a specific node. Multiple monitoring nodes were used to do passive and active harvesting in opennet by logging all requests on said nodes and the identities[4] of neighbouring nodes. During a period of 80 hours, at any given time between 2,000 and 3,000 running opennet nodes were online and 11,100 unique identities were found[5]. They also concluded that it is

[2]The version of the FRED used in the experiments of Cramer et al. still used TCP. The exact version number was not mentioned in the paper.

[3]At the time of writing several solutions have been proposed, see `http://freenet.mantishub.io/view.php?id=3919` for details.

[4]Vasserman et al. uses the term pseudonym for what in Freenet is called the identity of a node.

[5]The research by Vasserman et al. [2009] was done on

impossible to know how many existing Freenet nodes there were in the network since nodes could have been offline during the experiment. Vasserman et al. chose to log identities instead of IP addresses as this should give a better indication of the amount of users in a network, since a node can rejoin the network with a different IP address while keeping the same identity.

A routing table insertion attack on opennet was done by Baumeister et al. [2012]. Based on the assumption that the topology is known in advance, it was shown that it is possible for a malicious node to connect to a random node in the network. Their method for obtaining the topology was based on the neighbour-of-neighbour sharing feature of nodes and using the experimental PROBE message[6]. A set of attacking nodes connects to regular nodes in the network and starts inserting and requesting files on paths that contain the target node. Opennet nodes will drop neighbours if they do not successfully answer a specific number of queries, this feature was used to manipulate the neighbour set of the target node. The attacker would make sure that his requests along the path containing the target node would all have a high success chance by requesting known files. The successful queries allow for less successful peers to be dropped in favour of a malicious node. In addition, they have demonstrated a traceback attack based on the FNPRejectLoop message. It allows for the discovery of the path a message with a specific UID has taken. This work was further extended by Tian et al. [2015].

Roos et al. [2014] applied a number of passive and active mapping techniques to Freenet in both the opennet and a hybrid network[7]. By doing passive measurements using a number of monitoring nodes they attempted to find out if the distribution of nodes in opennet follows the small-world model by Kleinberg [2000b]. Their darknet experiment was limited to the hybrid model where the monitoring node was connected to both a darknet and the opennet. Network size and user origin was mapped by logging the IP addresses of opennet neighbours of a monitoring mode and obtaining the geolocation of said nodes based on their IP address. The IP addresses of the seed nodes can be logged as well, this combined with the addresses of regular nodes allows for a blacklist to be made to block the Freenet opennet. The popularity of a key k was calculated by dividing the number of requests for k by the total number of requests. An active measurement was done using probes that are part of the FNP. Two probes: FNPRoutedPing and FNPRHProbeRequest were used to get information on the availability of nodes. The former probe has been disabled by default in September 2012 due to privacy concerns. In the opennet monitoring experiments they found close to 60,000 unique nodes during a period of 14 days. In addition to their experiments, Roos et al. criticised the Freenet white papers for basing their conclusions on simulated network topologies. They claim these are unrealistic as they do not take into account uniform join-leave churn, consider all nodes to have the same storage capacity and consider all content to be equally popular. The research done by Roos et al. employed either opennet and hybrid networks, but no work was done on a pure darknet.

A traceback attack was developed by Tian et al. [2015] for opennet. The location of a suspect node is determined using the routing table insertion attack described in Baumeister et al. [2012]. By connecting to a suspect node with an attack node the former can be queried to check if it has seen a message with a specific UID. With 24% to 43% accuracy a node can be traced back which had send a specific content request message. This attack could also be applied to darknet, but the higher level of trust needed for connecting to the suspect node might make it more difficult.

4 Method

The experimental setup is outlined in Section 4.1. Section 4.2 describes the method used to detect

Freenet 0.7 build #1204 r25665 (2-17-2009).

[6]It is unclear which command it exactly corresponds with in the current build of the FRED.

[7]Roos et al. [2014] used a number of different versions of Freenet during their research: Freenet 0.7 build #1407, #1410, #1442 and #1457.

Freenet traffic based on its characteristics. The details of the implementation can be found in Section 4.3, and the data collection is explained in Section 4.4.

4.1 Experimental setup

The FRED version 0.7 build #1477 (2017-03-09)[8] was used for the experiments. A Xen hypervisor was set up with Ubuntu 16.04 LTS as host *Operating System* (OS). The nodes were installed on separate *Virtual Machine*s (VMs) running Ubuntu 16.04 LTS with sufficient resources to run a Freenet node. The specifications of the physical machine on which the experiments were done is given in Table 1 and the specifications of the VMs is given in Table 2. The nodes were assigned IPv4 addresses in the same subnet. The VMs were connected to each other via a bridge, which was also used to capture the traffic on. The assumption is that the attacker is able to tap the network traffic of the machines he suspects might be running Freenet nodes, like possible in a company network or at an *Internet Service Provider* (ISP). During the experiments the VMs had no connection to the Internet.

The Freenet nodes were set up with both the physical threat and the network threat level to "HIGH". The friends[9] were added with trust set to "LOW", and as secret contacts, meaning no one in the network except the two nodes know about their peering. In higher trust levels the nodes will share information about their neighbours to increase routing performance. These settings should ensure that the nodes run at the highest possible level of security while having minimum trust in their peers. The combination of settings should result in as little information leakage as possible, making detection as difficult as possible. The storage capacity was set to 512 MiB. No bandwidth limitation was set, while the download speed was set to 384 KiB/s, and the upload speed to 16.0 KiB/s. All other settings were kept at their default value as set by the installation wizard in the FProxy. The FProxy is the *Graphical User Interface* (GUI) which allows the operator to change settings of his node. The topology of the darknet used in the experiments can be seen in Figure 4. Each node has a degree of at least three, which concurs with the instructions on the FProxy installation wizard to have at least three friends online at any given time.

[8]Commit d38a867b12ab2e245b33f4a3ca986ccdc13297d9 of `https://github.com/freenet/fred.git`.

[9]Peers are called "friends" in Freenet.

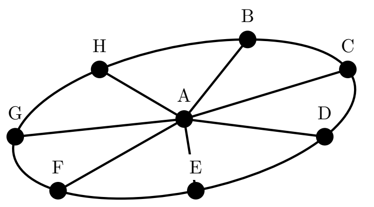


Figure 4: Topology of the darknet training setup.

Table 1: Server specification.

Hypervisor	Xen version 4.6.5
Host OS	Ubuntu 16.04 LTS
Kernel	GNU/Linux 4.4.0-79-generic x86_64
CPU	Intel® Xeon® CPU E3-1240L v5
RAM	16GiB @ 2133MHz

4.2 Detection

The detection of Freenet nodes is done in two steps. First, the traffic is matched to the characteristics of FRED traffic as explained in Section 4.2.1. Second, the traffic is compared to a baseline by testing it on an *Support Vector Machine* (SVM) that has been trained on Freenet traffic as described in Section 4.2.2.

Table 2: VM specification.

OS	Ubuntu 16.04 LTS
Kernel	GNU/Linux 4.4.0-79-generic x86_64
vCPUs	2
Memory	1024 MB
Swap	1024 MB
Storage	10 GB

Table 3: The composition of a typical IPv4 packet as sent by the FRED. The UDP payload consists of the *Hash-based Message Authentication Code* (HMAC), one or more fragments, and padding.

Bytes	Description	
20	IPv4 header	
8	UDP header	
10	HMAC	
10-1222	*Fragments*	
	Bytes	**Description**
	1-9	Fragment header
	1 >	Fragment data
0-127	Padding	

4.2.1 Step #1: Traffic characteristics

The following section explains the characteristics of Freenet traffic. The findings have been based on a review of the FRED source code [Freenet Project, 2017c], and instructions in the FRED FProxy.

A number of values are hard coded into the FRED to calculate the maximum and minimum IP packet length. When the term IP packet length is used in this paper, the length of the whole IP packet is meant, including the IP header. The *Maximum Transmission Unit* (MTU) is set to 1280 bytes, and the IPv4+UDP header length is set to 28 bytes while the IPv6+UDP header is set to 48 bytes. The FRED conservatively chooses the latter for the calculation of the maximum UDP payload, when it was unable to detect which IP version is used. As it was unable to do so in the experimental setup, this leaves a maximum UDP payload of 1232 bytes for both IPv4 and IPv6. See Appendix A for an explanation featuring examples from the FRED source code. An HMAC with a length of 10 bytes is added to the UDP payload (see Appendix B for details), leaving 1222 bytes for the actual messages. The UDP payload is padded to the nearest multiple of 64 bytes with an extra random 0 to 64 bytes [Freenet Project, 2006], and is encrypted with *Advanced Encryption Standard* (AES) in *Periodic Cipher Feed Back* (PCFB) mode.

The nodes send messages to their peers as part of the FNP. The maximum message size is 4096 bytes to prevent big messages from starving out other messages resulting in timeouts, which is even more of a problem when retransmits happen (see Appendix C for details). A message can have submessages which get parsed simultaneously and each have their own separate maximum message size.

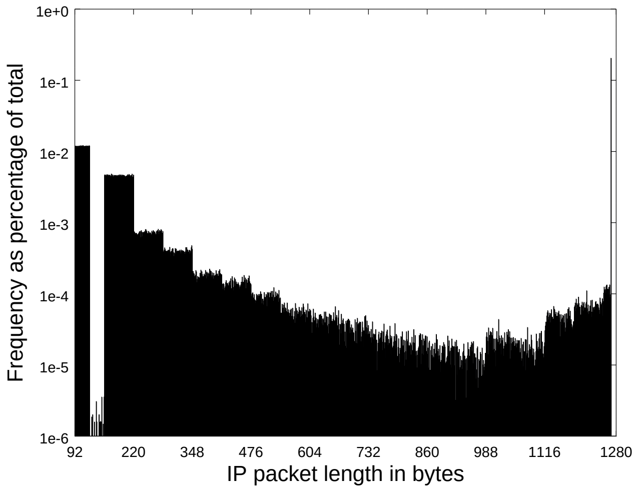
The UDP payload contains one or more fragments. A fragment can correspond with a complete single message or a part of it in the case of large messages. Each fragment has a minimum size of 1 byte and a maximum header size of 9 bytes. The minimum fragment size is 10 bytes, see Appendix D for details. Thus, the minimum unpadded UDP payload is 20 bytes: the HMAC plus a single minimum fragment. The composition of a typical packet as sent by the FRED in the experimental setup can be seen in Table 3.

The FRED will use the same UDP port to receive messages from all its peers. The port is chosen at random during the installation, but stays the same during the life time of a node. It should be common for a node to have at least three different source addresses that send packets to the same UDP port as the FProxy installation wizard recommends a user to get at least this amount of active peers.

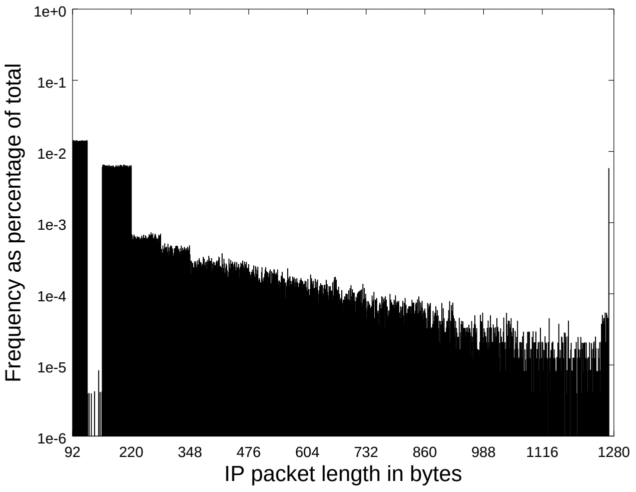
The findings can be formulated in a set of rules used to determine if a machine is transmitting traffic that is characteristic for the FRED. For this experiment only IPv4 will be considered and traffic with the default padding enabled.

1. Port number between 1024 and 65535.
2. Maximum IP packet length of 1280 bytes.
3. Minimum IP packet length of 92 bytes.
4. Maximum UDP payload of 1232 bytes.
5. Minimum UDP payload of 64 bytes.
6. An IP address receiving packets on the same UDP port from at least three different IP addresses.
7. A socket has to have sent and received at least one packet.

Some initial packet captures were done to verify the rules. Figure 5a displays the average normalised



(a) Nodes being queried.



(b) Idle.

Figure 5: Average of the normalised IP packet length frequencies of the Freenet nodes in the test network for three hours. Note that the y-axis has a logarithmic scale.

packet length frequencies of the Freenet nodes in the test network while processing queries, and Figure 5b shows the network being idle (no inserts or requests by a user). Note that the maximum packet length found in the packet captures is 1260 bytes, not 1280 bytes, due to the 20 bytes overhead caused by the FRED not recognising the correct IP version that was used.

When a packet is found from a socket (meaning the four tuple consisting of: IP source address, IP destination address, UDP source port, UDP destination port) that does not comply with the above rules, the socket is flagged as not suspicious. All remaining sockets are marked as suspicious and will be under further investigation in step #2 in Section 4.2.2.

4.2.2 Step #2: Comparison to baseline

The steps performed in Section 4.2.1 delivered a list of suspicious sockets as well as a list of packets sent to those sockets. The traffic must then be validated against known Freenet traffic. This should be done to rule out that these sockets are not other applications which just happen to have stayed within the bounds of Freenet traffic. Since Freenet is a P2P file sharing network, its traffic might be similar to other P2P networks such as BitTorrent [Cohen, 2008].

A baseline of packet length frequencies was established per socket by doing a packet capture on the bridge connecting the VMs. Each socket thus is a data point, with each node having a number of data points equal to its degree. The packet length frequencies were normalised by dividing them by the total amount of packets that were sent via that specific socket. Every 10 minutes an insert was done on each node with between 1 and 10 blocks of 32 KiB of random data. The inserted file was then requested from a random other node. Next, a random non-existing file was requested from each node. These actions should correspond with the basic operations that are performed in Freenet. The topology remained static for the duration of the experiment, e.g. no nodes left or joined the network.

A traffic capture of 5.5 hours was done on the test network to establish a baseline. The normalised packet length frequency distribution of it was used to train an one-class SVM. An SVM is a supervised machine learning algorithm that is used for classification and regression problems. By applying kernel functions it allows for cheap computations of data in high-dimensional space. Please see Smola and Schölkopf [2004] for a concise explanation of the workings of an SVM.

The result of step #1 and step #2 should be a list of sockets which should have a high probability of being used by a Freenet node. This list can then be used to create a mapping of the network by creating a graph where each node corresponds with an IP source address or IP destination address in the list, and edges are drawn between the IP source and destination addresses of sockets. Note that it could be that the darknet found is a subgraph of a larger darknet which would have nodes with IP addresses outside of the captured traffic.

4.3 Implementation

Tcpdump (version 4.9.0) [Jacobson et al., 2003] was used to capture the traffic on the bridge. The packet captures were parsed with Tshark (version 2.2.6) [Combs, 2012] to extract only the UDP packets, the sockets, and the packet lengths. A Python script was written to apply the rules from Section 4.2.1 to the parsed captures. For the SVM the implementation of the scikit-learn Python package was used [Buitinck et al., 2013]. The *Radial Basis Function* (RBF) kernel was used with $\gamma = \frac{1}{1280-92}$ and $\nu = 0.5$, these were the default values as set by the scikit-learn package for this classifier. Changing them did not lead to an overall increase in accuracy.

4.4 Data collection

In order to test the classifier system with traffic generated from different sources and networks a number of sets were gathered. The test darknet was used to generate a set with the traffic generation mentioned in Section 4.2.2 and a set with the nodes idling. An overview of the data sets can be seen in Table 4. Set #1 was used to train the SVM. Set #2 and #3 were created at different times to have separate test cases for the training model. These captures contain all the UDP traffic as captured from the bridge connecting the VMs. In addition, a number of packet captures were done on applications that used P2P and UDP to see if any false positives would arise. Capture set #4, #5 and #6 were done on a desktop computer running Ubuntu 16.04 LTS and only contain traffic from the specified UDP ports the programs used.

5 Results

Table 5 shows the percentage of true positives and the number of false positives found after step #1. In the second step 4-fold cross-validation was done, Table 6 shows the mean sample score and the standard deviation. When no suspicious sockets were found in step #1, there would be no data to test the SVM on. In this case the corresponding cell for step #2 has been filled with black.

6 Discussion

No false positives were found in the data sets of the other P2P applications. In the case of BitTorrent a considerable number of IP packets were smaller than the minimum packet size of 92 bytes. Most IP packets lengths had the MTU size of the interface, which was higher than the maximum packet length allowed by Freenet. In the case of OpenArena and traceroute there were many IP packets with a length smaller than 92 bytes. The filtering step was so effective that it was difficult to find any applications that could pass it. The list of P2P applications tested is by far not exhaustive and it could be that there are programs which could pass the filtering step[10].

The accuracy on the idle data set was not comparable to the data set with traffic. The lower frequency at 1260 bytes caused by the lack of user queries prevented the SVM from classifying the traffic correctly. As part of the FNP, files would be reinserted into the

[10]However, the author was not able to find any.

Table 4: Overview of the data sets.

Set	Freenet sockets	Time	File size	Description
1	28	5.5 hours	433 MB	Test network with traffic generation, with the first 30 minutes idling.
2	28	3 hours	247 MB	Test network with traffic generation.
3	28	3 hours	55 MB	Test network idling.
4	0	5 minutes	461 MB	Traffic from the BitTorrent client Transmission (version 2.84) while using the following torrent file: `http://releases.ubuntu.com/17.04/ubuntu-17.04-desktop-amd64.iso.torrent`.
5	0	5 minutes	1 MB	OpenArena client (version 0.8.8-15/Ubuntu), while refreshing the Internet server list two times and then participating in an online game. The P2P protocol used in the game is based on the Quake 3 protocol.
6	0	1 minute	39 KB	Traceroute (version 2.0.21) while running on port 2000. Four traceroutes were done for each of the following domains: `os3.nl`, `google.com`, and `uva.nl`.

Table 5: The number of true positives and false positives in step #1.

Set	True positives	False positives
1	28	0
2	28	0
3	28	0
4	0	0
5	0	0
6	0	0

Table 6: The mean score and standard deviation of the 4-fold cross-validation done in step #2.

Set	$\bar{x}$	s
1	75%	12%
2	43%	17%
3	14%	10%
4		
5		
6		

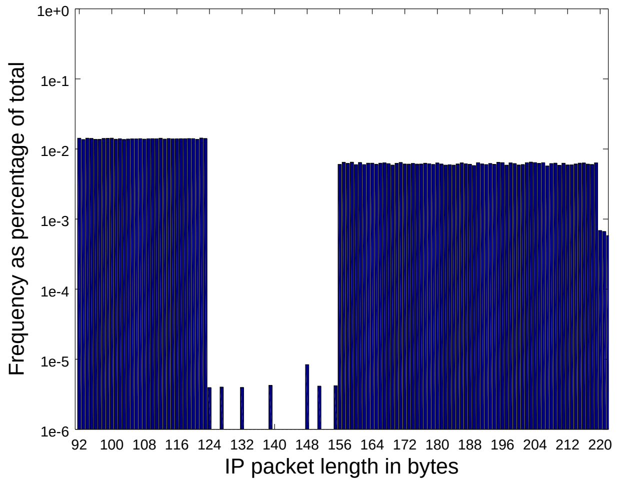


Figure 6: Average of the normalised packet length frequencies of the Freenet nodes in the test network for three hours while idle. Only the first 130 packet lengths are shown. Note that the y-axis has a logarithmic scale.

network to prevent them from being lost, even while the network was idle. This number of inserts is still considerably less than in the data set with explicit traffic generation.

Figure 6 shows Figure 5b with the x-axis adjusted to display the drop in frequency of the packet lengths between 124 and 156 bytes. The phenomenon occurs in both the idle network as well as in the network with requests and inserts. The reason for the small amount of packets with these specific lengths is unclear.

In the case of testing on the training set an accuracy was achieved of 75%. Since the SVM was fitted on the same data, it would be expected that an accuracy approaching 100% would have been achieved here. It is unclear why this happened, but it might have to do with the small data set. The low accuracies and high standard deviations for set #2 and #3 could be caused by the one-class SVM not being suited for this type of classification problem. Other machine learning algorithms were also tested: Naïve Bayes and Random Forest, but these did not score as high as the SVM. The usage of a small network had lead to very few data points which might have also impaired the working of the SVM. A larger data set could be created by adding more nodes and more connections to the network since each socket (connection) corresponds with a data point. Future work could look into testing the system with a larger data set, and possible other machine learning algorithms, to see if a higher accuracy can be achieved.

The experiments were done on a very small network and in unrealistic conditions. The nodes in the experimental darknet were all homogeneous. It would be unlikely in real life that users all have access to identical machines and Internet connections to run their nodes on. In addition, the simulation did not take into account the leaving and joining of nodes in the network. Users of Freenet usually do not have their node running constantly [Cramer et al., 2004]. Most likely they would also not be making a request or insert consistently every ten minutes, but only in bursts when their node is online and they are actively using it.

A node might disconnect from the network and later rejoin with a different IP address. This would result in the node being "found" multiple times, as there is no way to differentiate between the instances. One could use the source port and peer information to indicate an overlap, but this would not give hard evidence that the two IP addresses belong to the same node. Combining Freenet with *The onion router* (Tor) would result in such a situation, as the different Tor exit nodes would have distinctive IP addresses.

The traffic analysis and experiments were all done on the FRED. Although this is the most commonly used daemon to communicate in Freenet, it should be noted that there might be other implementations out there which might have different traffic patterns. More specifically, members of a darknet communally could agree to patch their nodes in a number of ways. First, they could set a higher MTU. Second, use a lower minimal padding than 64, this would thus also lower the minimum packet size. Third, use a port number lower than 1024, even though this would require Freenet to run in privileged mode, which is not recommended for security reasons. These actions would allow them to evade the rules in step #1 of the detection and possible also allow their traffic to be misclassified by the SVM[11].

The detection of Freenet darknet nodes as presented in this paper can be scaled up, given an attacker has enough resources. The first step would filter out almost all non-suspicious traffic, leaving very little work to be done by the SVM (which is computationally more expensive). It can be applied to small networks as presented in this paper, at an ISP or even at a national level. The latter combined with the blacklisting of opennet (seed) nodes is especially worrisome since it would mean that Freenet can be blocked and that it can be censored in any of its topology types. In the *People's Republic of China* (PRC) Freenet version 0.5 has been blocked [Freenet Project, 2017e], like other anonymisation networks such as Tor and *The Invisible Internet Project* (I2P). In order to pre-

[11]The port number is not a feature used by the SVM, so changing this would have no effect on the classification.

vent censorship from taking place at darknet level the Freenet project should take steps into the direction of making the traffic and nodes harder to detect. An attacker with a significant amount of resources could even do a mapping of the hybrid Freenet. An opennet mapping would first be done to get an initial set of IP addresses of nodes. These could then be traced back to their ISP and an ISP level darknet mapping might reveal any additional darknet connections that these nodes have. Although the method presented in this paper does not have a 100% accuracy, merely the suspicion that someone might be running a Freenet node could warrant for further research by law enforcement agencies. Some suggestions for improving the design of Freenet to become an MCON are mentioned in Vasserman et al. [2009]. Section 8 gives some future work that could be done to look into the mitigation of the attacks discussed in this paper.

The lack of masking Freenet nodes and traffic has been known by the core developer team since early in the development: "Making nodes invisible is not easy by any stretch of the imagination and is not something we can or should address before 1.0" [Clarke and Toseland, 2005]. The FProxy installation wizard notes the following: "Freenet cannot hide connections made to others, only what is transferred through them" [Freenet Project, 2017a]. At the time of writing, there is no clear mention in the project roadmap for implementing any type of concealment feature [Freenet Project, 2017b]. This seems to clash with the goal of the Freenet project to create a truly membership-concealing censorship resilient overlay network.

7 Conclusion

An overview has been given of Freenet opennet attacks, mapping techniques, and their applicability to the darknet. Due to the continuous development of the Freenet project, findings in the past cannot always be replicated in the present. In addition, most attacks for the opennet cannot be applied to the darknet due to the different nature of the two topologies.

A two step method for the detection of Freenet darknet nodes has been introduced, which can retrieve the IP address of a Freenet node based on captured traffic. In an idle network, and a network that is doing random inserts and requests, the mean accuracy is respectively 14% ($s = 10\%$) and 43% ($s = 17\%$). The interconnectivity of the retrieved nodes can be detected based on the traffic, which allows an attacker to construct a mapping of the darknet.

8 Future work

One could consider evading the SVM trained on the normalised IP packet length frequencies by padding all packets to the same size. This approach has been taken by Tor, but traffic analysis remains possible based on the frequency and direction of packets [Juarez et al., 2014]. A straightforward application for Freenet would be to always pad all packets to a random specific length, or to the maximum MTU. The implementation of the latter can be seen in Appendix E. There would be a significant increase in overhead, while applying rule based detection on the traffic would still be possible by creating a rule to match for this specific length. In the case of a random length it could be made so that each node has a random unique length it would pad to. For example, node A might always pad to 1337 bytes, while node B will pad to 1290 bytes. Traffic analysis would be required by an attacker to discover the specific length for each node before a detection rule could be made.

By adding extra logging rules to the FRED source code it should be possible to determine size of the outgoing packets and which message types they contain. This information could then be used to make assumptions on which message types were sent based on IP packet lengths as found in traffic from Freenet nodes. Since a packet may contain multiple messages, and due to the padding, it might be a difficult task to achieve. Future work could look into the feasibility of creating such a mapping and any methods to prevent it from happening. The latter could be done by padding all the packets to the same size. This would make it impossible to distinguish between the different message types based on the length of the packets.

When data is inserted which is larger than the MTU, multiple packetTransmit messages will be sent. In the case of large files this would lead to a large number of these types of messages being sent, which would almost all have the maximum IP packet size. This corresponds with the higher frequency at 1260 bytes in Figure 5a and Figure 5b. When a capture is done of the entire network, this might be used to trace the node which did the original insert. Since the network occasionally reinserts data, there might be a overlap between new files being inserted and reinserts, this might make the task difficult. Future work could look into the feasibility of this attack.

The experiments were all done using IPv4, while the FRED also supports IPv6. The rules in Section 4.2.1 should also apply for IPv6 in the test network since the FRED takes into account the larger overhead of IPv6 UDP traffic, regardless of IP version used (see Appendix A for details). It is unclear if an SVM trained on IPv4 traffic would also detect IPv6 Freenet nodes, or even a mixture of both. IPv6 traffic should have a larger average IP packet length due to the increase in the IP header size. Future research could look into training a data set on a mixture of both IP versions and checking if the results found in this paper would also apply for mixed data sets.

9 Acknowledgements

The author would like to thank Yonne de Bruijn and Lennart Haagsma from Fox-IT for supervising this research project.

10 Attributions

Figure 2 has been based on Menge [2009].

Acronyms

AES *Advanced Encryption Standard*. 7

CHK *Content Hash Key*. 2

DHT *Distributed Hash Table*. 2

F2F *friend-to-friend*. 1

FCP *Freenet Client Protocol*. 4

FNP *Freenet Protocol*. 2, 4, 5, 7, 9

FRED *Freenet REference Daemon*. 2, 4–8, 11–13, 16, 17

GUI *Graphical User Interface*. 6

HMAC *Hash-based Message Authentication Code*. 7, 17

I2P *The Invisible Internet Project*. 11

IDS *Intrusion Detection System*. 4

IP *Internet Protocol*. 1, 2, 4, 5, 7–9, 11–13, 16

ISP *Internet Service Provider*. 6, 11, 12

KSK *Keyword Signing Key*. 2

MCON *membership-concealing overlay network*. 4, 12

MTU *Maximum Transmission Unit*. 7, 9, 11–13, 16, 18

OS *Operating System*. 6

P2P *peer-to-peer*. 1, 4, 8–10

PCFB *Periodic Cipher Feed Back*. 7

PRC *People's Republic of China*. 11

RBF *Radial Basis Function*. 9

SSK *Signed Subspace Key*. 2

SVM *Support Vector Machine*. 6, 9, 11–13

TCP *Transmission Control Protocol*. 2, 4

Tor *The onion router*. 11, 12

UDP *User Datagram Protocol*. 2, 4, 7–9, 13, 16

UID *Unique Identifier*. 2, 5

USK *Updateable Subspace Key*. 2

VM *Virtual Machine*. 6, 8, 9, 16

References

Todd Baumeister, Yingfei Dong, Zhenhai Duan, and Guanyu Tian. A routing table insertion (rti) attack on freenet. In *Cyber Security (CyberSecurity), 2012 International Conference on*, pages 8–15. IEEE, 2012.

Peter Biddle, Paul England, Marcus Peinado, and Bryan Willman. The darknet and the future of content protection. In *ACM Workshop on Digital Rights Management*, pages 155–176. Springer, 2002.

Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

Ian Clarke. A distributed decentralised information storage and retrieval system. Master's thesis, University of Edinburgh, 1999.

Ian Clarke and Matthew Toseland. Freenethelp.org wiki, 2005. URL `http://www.freenethelp.org/html/AttacksAndWeaknesses.html`. Consulted on 2017-06-21. The page contains an informal discussion on attacks and weaknesses of Freenet. *Toad* is the pseudonym used by Matthew Toseland.

Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Designing Privacy Enhancing Technologies*, pages 46–66. Springer, 2001.

Bram Cohen. The bittorrent protocol specification, 2008. URL `http://www.bittorrent.org/beps/bep_0003.html`. Consulted on 2017-06-26.

Gerald Combs. Tshark - dump and analyze network traffic, 2012. URL `https://www.wireshark.org/`. Consulted on 2017-06-29.

Curt Cramer, Kendy Kutzner, and Thomas Fuhrmann. Bootstrapping locality-aware p2p networks. In *Networks, 2004.(ICON 2004). Proceedings. 12th IEEE International Conference on*, volume 1, pages 357–361. IEEE, 2004.

Nathan S Evans, Chris GauthierDickey, and Christian Grothoff. Routing in the dark: Pitch black. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pages 305–314. IEEE, 2007.

Freenet Project. Freenet mantishub bugtracker issue n. 0000245., 2006. URL `https://freenet.mantishub.io/view.php?id=245`. Consulted on 2017-06-21.

Freenet Project. Freenet mantishub bugtracker issue n. 0005294., 2011. URL `https://freenet.mantishub.io/view.php?id=5294`. Consulted on 2017-07-06.

Freenet Project. Freenet mantishub bugtracker issue n. 0005678., 2013a. URL `https://freenet.mantishub.io/view.php?id=5678`. Consulted on 2017-07-06.

Freenet Project. Freenet mantishub bugtracker issue n. 0005679., 2013b. URL `https://freenet.mantishub.io/view.php?id=5679`. Consulted on 2017-07-05.

Freenet Project. Freenet mantishub bugtracker issue n. 0000976., 2013c. URL `https://freenet.mantishub.io/view.php?id=976`. Consulted on 2017-07-06.

Freenet Project. Freenet wiki - fproxy, 2017a. URL `https://github.com/freenet/wiki/wiki/FProxy`. Consulted on 2017-06-26. Page contains information on how to access the FProxy.

Freenet Project. Freenet mantishub bug-tracker roadmap, 2017b. URL `http://freenet.mantishub.io/roadmap_page.php`. Consulted on 2017-06-26.

Freenet Project. Freenet reference daemon source code, 2017c. URL `https://github.com/freenet/fred.git`. Consulted on 2017-06-28.

Freenet Project. Freenet website, 2017d. URL `https://freenetproject.org/`. Consulted on 2017-06-19.

Freenet Project. Freenet wiki, 2017e. URL `https://github.com/freenet/wiki/wiki/`. Consulted on 2017-06-06.

Van Jacobson, Craig Leres, and Steven McCanne. Tcpdump public repository, 2003. URL `https://github.com/the-tcpdump-group/tcpdump`. Consulted on 2017-06-29.

Marc Juarez, Sadia Afroz, Gunes Acar, Claudia Diaz, and Rachel Greenstadt. A critical evaluation of website fingerprinting attacks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 263–274. ACM, 2014.

Jon Kleinberg. Navigation in a small world. *Nature*, 406(6798):845–845, 2000a.

Jon Kleinberg. The small-world phenomenon: An algorithmic perspective. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 163–170. ACM, 2000b.

Andreas Menge. Latex tikz 3d graph model example, 2009. URL `http://www.texample.net/tikz/examples/3d-graph-model/`. Consulted on 2017-06-17.

Mohamed Othman and Mostafa Nikpour Kermanian. Detecting and preventing peer-to-peer connections by linux iptables. In *Information Technology, 2008. ITSim 2008. International Symposium on*, volume 4, pages 1–6. IEEE, 2008.

Stefanie Roos, Benjamin Schiller, Stefan Hacker, and Thorsten Strufe. Measuring freenet in the wild: Censorship-resilience under observation. In *International Symposium on Privacy Enhancing Technologies*, pages 263–282. Springer, 2014.

Alex J Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and computing*, 14(3):199–222, 2004.

Solarwinds. Solarwinds forum, 2017. URL `https://thwack.solarwinds.com/thread/77015`. Consulted on 2017-06-21.

Guanyu Tian, Zhenhai Duan, Todd Baumeister, and Yingfei Dong. A traceback attack on freenet. *IEEE Transactions on Dependable and Secure Computing*, 2015.

Eugene Vasserman, Rob Jansen, James Tyra, Nicholas Hopper, and Yongdae Kim. Membership-concealing overlay networks. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 390–399. ACM, 2009.

Appendix A MTU and IP packet size

Listing 1 shows the part of the FRED source code which defines the values for the minimum and maximum MTU for IPv4 and IPv6. The default IP+UDP header size is set to the IPv6+UDP value of 48 bytes. For the maximum UDP payload size the value is used from the innerCalculateMaxPacketSize method, where the minimum of MAX_ALLOWED_MTU and the result of the getMinimumMTU method is subtracted by UDP_HEADERS_LENGTH. During the experiments the result of innerCalculateMaxPacketSize always was 1232, as observed by the maximum UDP payload sizes seen in the packet captures. Thus, the value of the Math.min function on line 306 would have to be 1280 to get to the observed maximum UDP payload of 1232 bytes. This indicates that the IPv6+UDP header size is used as $1280 - 1232 = 48$.

Listing 2 shows the getMinimumMTU method which returns either the maximum IP packet size, or the MTU found by the IP detector. Since the minimum of MAX_ALLOWED_MTU and the result of the method always have been 1280, it can be concluded that the getMinimumMTU method always returns the maxPacketSize. This could be caused by one of two things: the value of "detected" is larger than or equal to "mtu", or the value of "ipDetector" is always "null". The MTU of the interfaces of the VMs was set to 1500, so the latter would have to be the case. The "ipDetector" variable was set to null because the correct IP version could not be detected. In addition, Java attempts to retrieve the MTU from the interfaces [Freenet Project, 2013b], which it was also not able to do in the experimental setup.

During the experiments only IPv4 was used, while the FRED was taking into account that the IP header would be 20 bytes larger, which lead to the observed maximum IP packet length of 1260 bytes. The end result is a wasted 20 bytes when IPv4 is used if the FRED is unable to detect that it is using this IP version.

A choice was made by the Freenet developers to have the MTU hard coded to prevent fragmentation of the IP packets [Freenet Project, 2011] [Freenet Project, 2013a]. An open issue is in the bugtracker which states the problem should be solved by doing a path MTU discovery [Freenet Project, 2013c].

Note that regardless of the MTU set by the node, the receiving end has a hard coded receive buffer of 1500 bytes as can be seen on line 194 of Listing 1. So the MTU has to be lower or equal to 1500 to allow the other side to parse the packets.

Listing 1: Selected lines from io/comm/UdpSocketHandler.java.

```
194 private static final int MAX_RECEIVE_SIZE = 1500;
267 // CompuServe use 1400 MTU; AOL claim 1450; DFN@home use 1448.
268 // http://info.aol.co.uk/broadband/faqHomeNetworking.adp
270 // http://www.studenten-ins-netz.net/inhalt/service_faq.html
271 // officially GRE is 1476 and PPPoE is 1492.
272 // unofficially, PPPoE is often 1472 (seen in the wild). Also PPPoATM is sometimes
    1472.
273 static final int MAX_ALLOWED_MTU = 1280;
274 static final int UDPv4_HEADERS_LENGTH = 28;
275 static final int UDPv6_HEADERS_LENGTH = 48;
276 // conservative estimation when AF is not known
277 public static final int UDP_HEADERS_LENGTH = UDPv6_HEADERS_LENGTH;
278
279 static final int MIN_IPv4_MTU = 576;
280 static final int MIN_IPv6_MTU = 1280;
281 // conservative estimation when AF is not known
282 public static final int MIN_MTU = MIN_IPv4_MTU;
```

```java

private volatile int maxPacketSize = MAX_ALLOWED_MTU;

/** Recalculate the maximum packet size */
int innerCalculateMaxPacketSize() { //FIXME: what about passing a peerNode though and doing it on a per-peer basis? How? PMTU would require JNI, although it might be worth it...
    final int minAdvertisedMTU = node.getMinimumMTU();
    return maxPacketSize = Math.min(MAX_ALLOWED_MTU, minAdvertisedMTU) - UDP_HEADERS_LENGTH;
}
```

Listing 2: Selected lines from node/Node.java.

```java
public int getMinimumMTU() {
    int mtu;
    synchronized(this) {
        mtu = maxPacketSize;
    }
    if(ipDetector != null) {
        int detected = ipDetector.getMinimumDetectedMTU();
        if(detected < mtu) return detected;
    }
    return mtu;
}
```

Appendix B HMAC length

Listing 3 shows the fixed HMAC length of 10 bytes as found in the source code of the FRED.

Listing 3: A line from node/NewPacketFormat.java.

```java
private static final int HMAC_LENGTH = 10;
```

Appendix C Maximum message size

Listing 4 displays the maximum message size and Listing 5 shows the comment explaining that large messages are bad for the traffic flow.

Listing 4: A line from node/NewPacketFormat.java.

```java
public static final int MAX_MESSAGE_SIZE = 4096;
```

Listing 5: Selected lines from node/MessageItem.java.

```java
buf = msg.encodeToPacket();
if(buf.length > NewPacketFormat.MAX_MESSAGE_SIZE) {
    // This is bad because fairness between UID's happens at the level of message queueing,
    // and the window size is frequently very small, so if we have really big messages they
    // could cause big problems e.g. starvation of other messages, resulting in timeouts.
    // (especially if there are retransmits).
    Logger.error(this, "WARNING: Message too big: "+buf.length+" for "+msg2, new Exception("error"));
}
```

Appendix D Fragment size

In Listing 6 shows the maximum fragment header size and the minimum amount of data bytes in a fragment. For each fragment, at least 10 bytes are reserved.

Listing 6: A line from node/NewPacketFormat.java.

```
while ((packet.getLength() + 10) < maxPacketSize) { //Fragment header is max 9 bytes, allow min 1 byte data
```

Appendix E Padding to the maximum

In Listing 7 the change can be seen for NewPacketFormat.java to pad all packets to the maximum MTU. Applying this should overwrite any settings the user has made concerning the use of padding.

Listing 7: Diff of NewPacketFormat.java between the change made for always using maximum padding, and commit d38a867b12ab2e245b33f4a3ca986ccdc13297d9 of `https://github.com/freenet/fred.git`.

```
diff --git a/src/freenet/node/NewPacketFormat.java b/src/freenet/node/NewPacketFormat.java
index 863eeb8..937d3e0 100644
--- a/src/freenet/node/NewPacketFormat.java
+++ b/src/freenet/node/NewPacketFormat.java
@@ -493,6 +493,7 @@ public class NewPacketFormat implements PacketFormat {
                                }
                        }
                }
+               paddedLen = maxPacketSize;

                byte[] data = new byte[paddedLen];
                packet.toBytes(data, HMAC_LENGTH, pn.paddingGen());
```