The Bitcoin Hunter: Detecting Bitcoin Traffic Over Encrypted Channels*

Fatemeh Rezaei[1][0000−1111−2222−3333], Shahrzad Naseri[2,3][1111−2222−3333−4444], and Amir Houmansadr[3][2222−−3333−4444−5555]

[1] College of Information and Computer Sciences, Amherst, MA 01002, USA
[2] Springer Heidelberg, Tiergartenstr. 17, 69121 Heidelberg, Germany
lncs@springer.com
http://www.springer.com/gp/computer-science/lncs
[3] ABC Institute, Rupert-Karls-University Heidelberg, Heidelberg, Germany
{abc,lncs}@uni-heidelberg.de

Abstract. Bitcoin and similar blockchain-based currencies are significantly important to consumers and industry because of their applications in electronic commerce and other trust-based distributed systems. Therefore, it is of paramount importance to the consumers and industry to maintain reliable access to their Bitcoin assets. In this paper, we investigate the resilience of Bitcoin to blocking by the powerful network entities such as ISPs and governments. By characterizing Bitcoin's communication patterns, we design classifiers that can distinguish (and therefore block) Bitcoin traffic even if Bitcoin traffic is tunneled through an encrypted channel like Tor and even if Bitcoin traffic is being mixed with background traffic, e.g., due to browsing websites. We perform extensive in-the-wild experiments to demonstrate the reliability of our classifiers in identifying Bitcoin traffic even despite using obfuscation protocols like Tor pluggable transports. We conclude that standard obfuscation mechanisms are not enough to ensure blocking-resilient access to Bitcoin (and similar cryptocurrencies), therefore cryptocurrency operators should deploy tailored traffic obfuscation mechanisms.

Keywords: keyword · keyword · keyword.

1 Introduction

Bitcoin and similar blockchain-based currencies [?] have seen rapid adoption by consumers and industry because of their many applications in electronic commerce and other trust-based distributed systems. Bitcoin supports \$1–\$4.2B worth of transactions per day, growing steadily.Bitcoin and similar virtual currencies offer significant advantages compared to traditional electronic currencies, which include open access to a global e-commerce infrastructure, lower transaction fees, cryptographically supported contracts [?] and services [?], and transnational operations.

* Supported by organization x.

Given this significant importance of electronic currencies, they need to be resistant to embargoes by governments. That is, people investing in cryptocurrencies (by running businesses that rely on such currencies) should be assured that their Internet providers or governments are not able to prevent them from using their cryptocurrencies if they decide too. For the sake of argument, consider what happens if the Great Firewall of China decides to block all Bitcoin traffic overnight.

In this paper, we investigate the resilience of Bitcoin to blocking by powerful network entities, including ISPs and governments. Note that identifying standard (non-encrypted) Bitcoin traffic is trivial as Bitcoin messages use specific packet contents and formats. Therefore, a trivial countermeasure to prevent an ISP from identifying Bitcoin traffic is to tunnel Bitcoin over an encrypting tool, e.g., VPN, SSH, or Tor. However, previous studies [**?**,**?**,**?**] show that encryption is *not enough* to conceal the nature or even the content of communications. Such attacks are broadly known as *traffic analysis*.

In this paper, we investigate if and how Bitcoin's traffic can be identified through traffic analysis despite being tunneled through an encrypted channel. First, we characterize Bitcoin's traffic patterns such as rates, timings, and sizes. Comparing with other protocols, we show that Bitcoin has traffic patterns that are unique, because of the specific types of messages sent by Bitcoin peers. Leveraging such unique features of Bitcoin traffic, we design a toolset of classifiers in order to distinguish Bitcoin traffic over encrypted channels. We perform extensive evaluations of our classifiers by capturing Bitcoin traffic in the wild. Particularly, we use more than two month of Bitcoin and other protocols traffic over Tor [**?**] and three Tor pluggable transports [**?**], namely, FTE [**?**], meek [**?**], and obfs4 [**?**] to evaluate our classifiers.

Our experiments show that while such obfuscation mechanisms modify Bitcoin's traffic by changing the sizes of packets, and changing packet latencies, for each of the protocols we are able to design a reliable classifier to identify Bitcoin traffic from other traffic. Our classifiers can even detect Bitcoin traffic mixed with background traffic such as open browser tabs.

Based on our experiments, we conclude that standard obfuscation mechanism do not do a good job in hiding Bitcoin traffic. This is due to a fundamental issue: Bitcoin (and all blockchain protocols we are aware of) generate of particular protocol messages with unique sizes and frequencies. To hide such unique patterns, an obfuscating protocol needs to apply significant cover traffic or apply large perturbations. The latter option has significant implications to the security of a cryptocurrency system.

In summary we make the following main contributions:

1. We evaluate Bitcoin's traffic and characterize the patterns of Bitcoin traffic such as its packet sizes and traffic shape. We compare Bitcoin's traffic patterns to other popular protocols showing its patterns to be unique.
2. Based on our characterization of Bitcoin traffic, we design a range of classifiers whose goal is to identify Bitcoin traffic despite being tunneled through

an encrypted channel (like Tor) and in the presence of background noise (e.g., open browser tabs).

3. Using several months of Bitcoin traffic and other protocols, we perform experiments to evaluate our classifiers. We evaluate our classifiers when Bitcoin traffic is tunneled over Tor and three Tor pluggable transports of FTE [?], meek [?] and obfs4 [?]. Our classifiers are able to identify Bitcoin traffic in all cases with only 10 minutes of traffic.

2 Background on Bitcoin

Bitcoin is the most popular cryptocurrency. It uses a decentralized, peer-to-peer architecture [?], where each peer (e.g., client) is identified by her unique public key. Bitcoin clients exchange money through Bitcoin transactions, which are broadcasted on Bitcoin's p2p network.

To prevent double spending and similar violations, Bitcoin uses a public ledger called the *blockchain* to store all Bitcoin transactions. The blockchain is a chain of *blocks*, where each block contains a set of transactions and a *proof-of-work*. A proof-of-work is a piece of data which is time-consuming and costly to generate. However, verifying the proof-of-work is easy. Each block is valid if and only if all of its transactions and its proof-of-work are valid. A verified block is broadcast on the network to update all peers' local ledgers.

Bitcoin's P2P Network. Bitcoin nodes form a full-mesh P2P network, and they connect to each other over unencrypted TCP connections. Each node can connect to up to 125 peers, where up to 8 of them are outgoing connections and the rest are incoming connections. A node stays connected to a neighbor until they restart or drop, in which case the node tries to replace them [?]. Blocks and transactions are propagated by gossip. To avoid DoS attacks, peers only forward valid blocks and transactions; invalid blocks are discarded. Bitcoin peers can be separated into two types: routable and non-routable. The former are capable of accepting incoming connections, and the latter are not, for example because they are behind a NAT or firewall. However, it is worth mentioning that the official `Bitcoind` software does not precisely split its functionality among routable and non-routable peers.

Bitcoin Protocol Messages. Bitcoin communications involve various protocol messages that are created by Bitcoin peers. We divide Bitcoin protocol messages into two classes: *synchronization messages*, which are used for propagating user addresses and transactions in the Bitcoin network, and *block-related messages*, which are responsible for disseminating Bitcoin blocks. We introduce major Bitcoin messages in Appendix A due to space constraints. Please refer to Table 5 for a full list of Bitcoin protocol messages.

3 Characterizing Bitcoin Traffic

In this section, we demonstrate the unique features of Bitcoin traffic. We will show that such unique traffic patterns of Bitcoin makes it reliably distinguishable from other protocols even despite encryption and mixture with background traffic. We will use our characterization to design classifiers for Bitcoin in the following sections. All of our experiments in this section follow the experimental setup and datasets described in Section 5.

Table 1: Number and percentage of each message in a 31 days Bitcoin traffic in compact block relaying

Message	Packets per minute	Proportion	Minimum packet size	Maximum packet size	Average packet size
`inv`	173.17	27.240%	67	1514	791.87
`getdata`	13.16	2.070%	68	1514	700.33
`block`	0.94	0.330 %	74	1514	772.74
`sendcmpct`	1.60	0.253%	63	1514	782.88
`cmpctblock`	2.02	0.319%	67	1514	789.21
`getblocktxn`	0.02	0.003 %	90	1360	367.67
`blocktxn`	0.77	0.121%	67	1514	777.16
`tx`	277.72	43.688%	66	1514	790.00

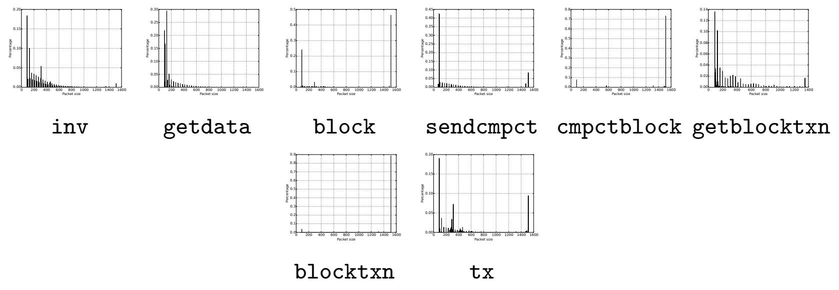
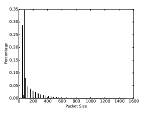


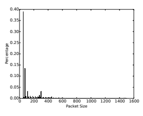
Fig. 2: Packet size distribution of Bitcoin communication messages in compact block relaying

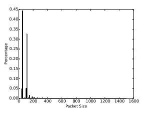
3.1 Proportion and Distribution of Messages

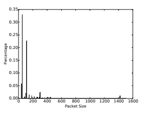
As discussed in the previous section, Bitcoin peers generate various kinds of messages. We show that the distribution and sizes of such messages are quite unique to the Bitcoin protocol, distinguishing Bitcoin traffic reliably from other protocols.

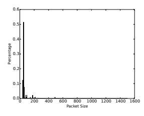
Table 1 demonstrates the proportion of different messages in the Bitcoin traffic we collected for 31 days. As can be seen, `tx` and `inv` are dominating with 43.6% and 27.2% of all packets, respectively. Therefore, the characteristics of these messages will shape the pattern of a Bitcoin peer's traffic.

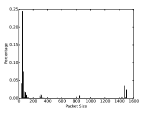

Compact block, upstream

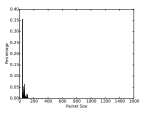

Compact block, downstream

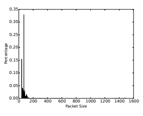

Full Block, upstream

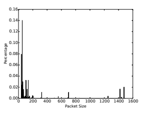

Full block, downstream

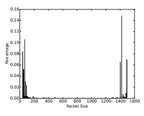

HTTP, upstream

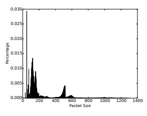

HTTP, downstream

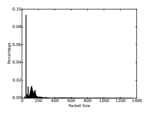

FTP, upstream

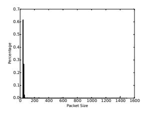

FTP, downstream


SSH, upstream


SSH, downstream


VoIP, upstream


VoIP, downstream


BitTorrent, upstream

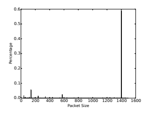

BitTorrent, downstream

Fig. 4: Upstream and downstream packet size distribution of Bitcoin and several popular protocols

Distribution of Packet Sizes. Figures 1a to 1h illustrate the packet size histogram of different types of Bitcoin messages in our collected Bitcoin traffic. As can be seen, each type of message has a distinguishing traffic pattern.

Histogram of packet sizes in aggregate traffic. Figures 3a and 3b show the histogram of packet sizes in the upstream and downstream directions, respectively, in compact block relaying. As mentioned before, `tx` and `inv` dominate the messages sent by a typical Bitcoin peer, therefore, their sizes (shown in Figures 1a and 1h) strongly shape the histogram of Bitcoin traffic, making them uniquely distinguishable from other protocols.

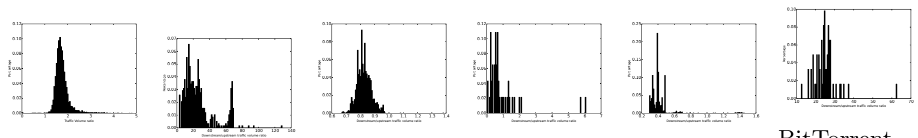
We also show the histogram of Bitcoin traffic in the full block relaying mode in Figures 3c and 3d. These histograms have a larger spike close to the MTU, unlike the case of compact block relaying. These are because of the larger block sizes (around 1MB) in the full block relaying.

Comparing to other protocols: Figures 3e to 3n show the histogram of other popular protocols, collected as described in Section 5. Note that we look at the traffic after going through an encryption tunnel, e.g., a VPN or SSH tunnel, so the histogram includes the (small) TCP ACK packets.

As we can see, the packet size distribution of Bitcoin is uniquely different from these other protocols, since a Bitcoin connection is composed of unique messages with specific size distributions shown before. For instance, the large number of `inv` messages shapes the overall distribution of sizes in Bitcoin traffic.

Ratio of Downstream to Upstream. We also measured the ratio of downstream to upstream traffic volume, which is shown in Figure 5a. Unlike other protocols like HTTP (shown in Figures 5b to 5f), Bitcoin traffic has a *symmetric* traffic volume in upstream and downstream. This is due to the fact that Bitcoin peers broadcast most of the bulky protocol messages they receive such as block and transaction announcements.

3.2 Shape of Traffic



Bitcoin, ratio per 5 minutes traffic

HTTP, ratio per website

FTP, ratio per 5 minutes traffic

SSH, ratio per 1 minute traffic

VoIP, ratio per 5 minutes traffic

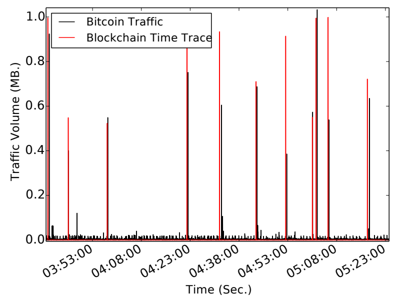
BitTorrent, ratio per 5 minutes traffic

Fig. 6: Ratio of downstream to upstream traffic volume

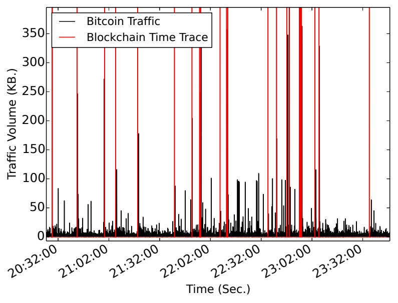
Above we showed that the counts and sizes of packets in Bitcoin demonstrate a unique behavior. In addition to that, the shape of traffic in Bitcoin and the volume of traffic received over the time is distinguishable from other protocols.

Full block relaying mode Figure 7a shows the traffic of a Bitcoin client operating in the full block relaying mode. As can be seen, the small protocol packets, mostly corresponding to `inv` and `tx` messages, appear uniformly over the time. On the other hand, the Bitcoin full blocks appear as large spikes of roughly 1MB at specific points in time, i.e., once a new block is generated in the network.

Compact block relaying mode In the compact block relaying mode, it would be harder to notice the block spikes, since only a sketch of the blocks is transmitted. In this mode, transmitting a compact block in the network results in smaller spikes of 100KB. Spikes of such small sizes may also occur when unverified transactions are transmitted, which will increase the detection's false positive. Also, a Bitcoin client may operate in the high bandwidth mode, in which the receiver node asks its peers to send new blocks without asking for permissions first. This will lead to more than one peer sending the same block at the same time. This and the large volume of missed transactions result in having spikes with more than 100 KB in the traffic. Figure 7b illustrates when and how compact blocks appear on a peer's traffic. As can be seen, compact blocks appear at smaller amplitudes than the actual block size, but the behavior is also nondeterministic, since it depends on whether the client has previously received some of the transactions in that block. This intuitively makes detection of compact blocks less reliable than full blocks, as shown later our experiments.

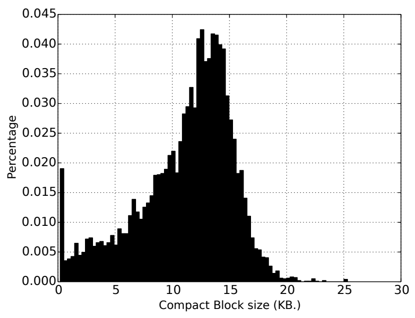


Full block

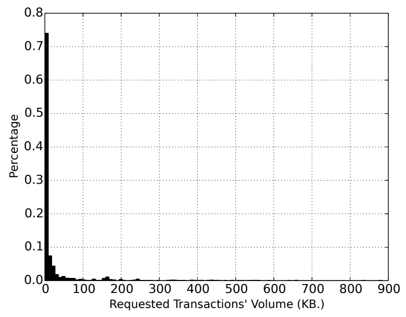


Compact block

Fig. 8: Comparing time of each block receive with time of blocks in the block chain in a) Full block and b) Compact block modes



Histogram of the size of compact blocks



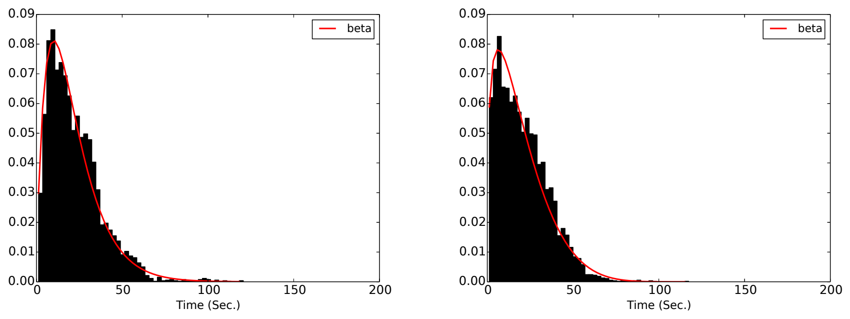
Histogram of the size of missing transactions

Fig. 10: Histogram of compact block mode components

We also measure the size of compact blocks by measuring the *length* field of `cmpctblock` messages, which is shown in Figure 9a. As can be seen, most of the compact blocks are as small as 15 KB (in contrast to 1MB full blocks).

We also measure the volume of transactions missing from an announced compact block (we do so based on the payload length of `blocktxn` messages). As described earlier, a Bitcoin client operating in the compact block mode will download such missing transactions. This is shown in Figure 9b. As we can see most of the transactions have volumes less than 100 KB.

Block propagation latencies. The propagation delay in the Bitcoin network is due to transmission delays and block verification by the receiving node at each hop. The transmission delay is the time to exchanging *inv* and *get data* messages, and sending the block via a *block* message.



Compact block

Full block

Fig. 12: Histogram of block propagation delay

We measure block propagation delay by subtracting the receiving time of the block message and the time stamp in the header of the block message. Figures 11a and 11b show the histogram of propagation delay for 6000 blocks in compact block and full block relaying, respectively. As shown in the Figure, we can model this empirical data using a Beta distribution [**?**].

4 Designing Bitcoin Classifiers

We use the features described above to build robust classifiers for Bitcoin traffic. We aim for our classifiers to work even in the presence of encryption and background traffic, e.g., when the machine running Bitcoin is used for web browsing and runs other applications, or when the Bitcoin traffic is tunneled over Tor or VPN. For a target connection, our classifiers extract specific features from it, using which decide if the target traffic contains Bitcoin or not. Each of our classifiers extract certain features as introduced below. We evaluate the performance of our classifiers using false positive, true positive and accuracy.

4.1 Size-based Classifier

As noted in Section 3.1, the histogram of Bitcoin packet sizes has a unique pattern. Based on this, we designed classifiers to distinguish Bitcoin traffic from other protocols.

Size histogram classifier (`SizeHist`) This classifier correlates the packet size histogram of a target user's traffic with the packet size histogram of Bitcoin traffic captured by the adversary. By histogram of packet sizes we mean number of each packet size from 1 to MTU size. To do so, the classifier first divides the given traffic into upstream and downstream directions, then it calculates the histogram of packet sizes in each direction. The baseline (real-time Bitcoin traffic) is also divided in upstream and downstream directions.

In the next step, the classifier calculates the cosine similarity between the histogram of the target traffic and several Bitcoin traces in both upstream and downstream direction. Finally, to detect if the target traffic is Bitcoin, the classifier compares the average of correlation values with a threshold. If both of the upstream and downstream averaged correlation values are above their thresholds, the target traffic is detected as Bitcoin traffic.

Algorithm 1 Size histogram classifier

```
1: procedure SIZE HISTOGRAM CLASSIFIER
2:     $B_{dwn}, B_{up} \leftarrow$ Size histogram of a Bitcoin downstream and upstream traffic
3:     $V_{dwn}, Vup \leftarrow$ Size histogram of input downstream and upstream traffic
4:     $cor_{dwn} = \frac{B_{dwn} \cdot V}{|B_{dwn}|^2 \times |V_{dwn}|^2}$
5:     $cor_{up} = \frac{B_{up} \cdot V}{|B_{up}|^2 \times |V_{up}|^2}$
6:     return $cor_{dwn}, core_{up}$
7: end procedure
```

Tor-specific Classifier (`SizeTor`): Some tunneling systems modify the sizes of packets to prevent information leakage. In particular, Tor [**?**] reformats traffic into constant-sized segments called *cells*. However, as the size of a cell is smaller than the MTU of IP packets, depending on the volume of traffic, multiple cells can merge into one single packet. This makes the number of single-cell packets and multiple-cell packets different for different protocols tunneled over Tor. Our `SizeTor` classifier aims at detecting Bitcoin traffic tunneled over Tor based on the distribution of single-cell packets. As shown in Section 3, Bitcoin traffic consists of a large number of small-size packets (due to frequent `inv` messages). Tor will add padding to these small packets to form cells. Therefore, the ratio of single-size packets in Bitcoin over-Tor is larger that regular traffic over Tor, e.g., HTTP-over-Tor. Based on this, our classifier compares the ratio of single-cell packets to all packets; if the ratio is larger than a threshold, the connection will be flagged as a Bitcoin connection. Note that while our `SizeTor` classifier is tailored for Tor, it can be adjusted for other protocols that similarly change the size of packets.

Tailoring for Tor Pluggable Transports Through our experiments on Bitcoin traffic over Tor pluggable Transports, we noticed that fraction of packets which have a specific size other than cell-size (for example, size 721 in FTE) are much larger than this value in normal traffic. Furthermore, this value is much larger than the fraction of packets which are in cell size. Therefore, when implementing sizeTor on pluggable transports, we compute the fraction of packets in these specific sizes to differentiate between Bitcoin traffic and others.

Downstream to upstream traffic volume ratio (D2U) As we discussed in Section 3, the ratio of downstream to upstream traffic volume is unique in Bitcoin. This classifier looks at t window of Bitcoin traffic. The t parameter should be at least 10 minutes to include at least one block in it. Then the downstream to upstream ratio is calculated for the target traffic. If the calculated ratio is larger than a threshold, which is determined empirically, the traffic will be detected as Bitcoin traffic. The threshold is defined in a way to distinguish Bitcoin traffic from other traffic while minimizing false positive. This detection scheme will be effective even in the situation that the underlying system changes the packet sizes and packet timings, since the transmitted traffic volume in both directions would stay the same.

4.2 Shape-based Classifier

We demonstrated in Section 3.2 that the shape of Bitcoin traffic is distinct from other protocols. We therefore design classifiers that try to identify (encrypted) Bitcoin traffic based on its shape. The main intuition of our shape-based classifier is looking for changes in the traffic volume of a target user around the times of block announcements. Therefore, we assume that the classifier obtains the times and sizes of Bitcoin blocks, e.g., from the public `blockchain.info` repository, or even by running a local Bitcoin client.

Algorithm 2 illustrates our window shape correlation algorithm. For each confirmed Bitcoin block, the algorithm analyzes the volume of the target traffic during the broadcasting time of that block. To do so, the algorithm defines two time windows with size ω_i around the block time, one before $(t_{block_i} - \omega_i, t_{block_i})$ and one after the block time $(t_{block_i}, t_{block_i} + \omega_i)$. The size of the window depends on the size of the block, the target client's bandwidth, etc., as evaluated later. Then, the algorithm evaluates the change in traffic during the two time intervals. For an actual Bitcoin traffic with no noise, the difference should be close to the size of the block. Therefore, the algorithm evaluates the difference of traffic volumes in the two consecutive windows and compares it to a threshold determined by the natural network jitter of the target (as discussed later). If the difference is within the bound, the algorithm considers that block to be *detected* in the traffic of the suspected user. The algorithm performs the same for a number of blocks and evaluates the ratio of such "detected" blocks. If the ratio is above a threshold η, the target user is declared to be a Bitcoin client.

Choosing the threshold η. The threshold should be chosen based on the target user's specific network conditions such as background traffic, network noise, and bandwidth. Therefore, the algorithm tailors the η parameter for each specific user.

To do so, the classifier generates N (e.g., $N = 100$) synthetic block series, which we call *ground false*s. Each ground false is generated based on the known pattern of Bitcoin blocks, e.g., by simulating blocks around 1MB roughly every 10 minutes for the full block relaying mode. More specifically, we generate the timing

Algorithm 2 Shape based classifier

```
 1: procedure Shape based Classifier
 2:     $B \leftarrow$ Block-chain time trace extracted from blockchain.info
 3:     $V \leftarrow$ Captured traffic volume in 1 second epochs
 4:     $N \leftarrow$ Total number of Blocks
 5:     for each block $b_i \in B$ do
 6:         $t_{b_i} \leftarrow$ Time of generation of block $b_i$
 7:         $||b_i|| \leftarrow$ Block's size
 8:         $\omega_i \leftarrow$ Window's size
 9:         $\Delta V_i = V(t_{b_i}, t_{b_i} + \omega_i) - V(t_{b_i} - \omega_i, t_{b_i})$
10:         if $||b_i|| - J \leq \Delta V_i \leq ||b_i|| + J$ then
11:             $detected\ block += 1$
12:         end if
13:     end for
14:     return $\frac{detected\ block}{N}$
15: end procedure
```

between blocks based on an Exponential distribution with mean 10 minutes. The classifier then correlates the target traffic with each of the N ground false instances using the correlation function of Algorithm 2. Finally, the threshold η is chosen as

$$\eta > \max(CF) \tag{1}$$

where CF is the set of correlation values against the N ground false instances. In another words, we choose η to be larger than the largest correlation value.

Choosing other parameters. The window shape classifier also needs to choose the values of the parameters ω and J. Parameter ω needs to be big enough to contain the most of the traffic of a block during block propagation. Moreover, Parameter J is chosen to take into account that some of the block propagation traffic might not be downloaded in that time window. This parameters needs to be selected based on user's bandwidth and the volume of background traffic , and therefore it needs to be chosen for each client specifically.

4.3 Neural Network-based Classifier (NN-based)

We also use Neural Network-based classifiers to detect Bitcoin in presence of a more complex background noise, e.g., browsing more than one website simultaneously. In following, we explain the feature selection phase, and then describe the design of our neural network.

Feature selection Again, we leverage the unique shape of Bitcoin traffic as discussed before to classify Bitcoin traffic using our NN-based classifier. To create each sample data, we divide time into intervals and use the volume of traffic in each interval as our features, which is presented in equation vlm $= v_1, v_2, ..., v_n$.

Note that v_I is the volume of traffic in interval I. We choose 10 minutes as the *sample size*, which is the smallest length to have at least one peak of traffic. Furthermore, to choose the interval length, we try different values of 1, 5, 10 and 20 seconds. From our experiments, we find out that the interval length of 10 seconds results in the best performance. Therefore, we choose 10 seconds as the interval length (l). Since the length of each sample is 10 minutes (600 seconds), using equation $n = \text{sample size}/l$, we get an array of length 60 as our feature vector.

Designing the model For our neural network model, we use a multi-layer perceptron that consists of an input layer, an output layer, and two fully-connected hidden layers. The input layer has $n = 60$ number of neurons, which is the size of each sample data, the hidden layers have $n_1 = 32$ and $n_2 = 16$ number of neurons, respectively, and the output layer has 1 neuron which represents if the sample data contains Bitcoin traffic or not. We use Relu as the activation function of the hidden layers and sigmoid [?] for the output layer. Also, we use binary cross-entropy as our loss function, and Adam optimization [?] to minimize the loss.

5 Experimental Setup

We use Bitcoin Core software[4] to run full node Bitcoin clients on multiple virtual machines on a campus network. Each virtual machine is connected to the Internet with high bandwidth. Before starting the experiments, we leave our Bitcoin clients for a few days to make sure they have downloaded an up-to-date blockchain ledger. We capture Bitcoin traffic under three different scenarios on a Linux 16.0.4 virtual machine:

5.1 Datasets

Collecting Bitcoin traffic: We use Bitcoin version 0.12.0 to capture Bitcoin traffic in the full block relaying mode and Bitcoin version 0.14.0 to capture traffic in the compact block relaying mode. We capture Bitcoin traffic for each version for a period of around a month. Specifically, we captured the Bitcoin traffic in the full block relaying mode from August 28th to October 9th, 2016, and Bitcoin traffic in the compact block mode from March 14th to April 18th, 2017.

Bitcoin tunneled through Tor We captured Bitcoin traffic behind Tor [?] for both compact and full block modes. We also captured Bitcoin traffic in the compact block mode behind Tor and popular Tor pluggable transports of obfs4 [?], FTE [?], and Meek-amazon [?].

Bitcoin with background traffic: We captured Bitcoin traffic in presence of HTTP background traffic by browsing the top 500 Alexa websites using the

[4] https://bitcoin.org/en/bitcoin-core/

Selenium[5] tool while running Bitcoin software. We also collected Bitcoin traffic with HTTP background for the same set of websites behind Tor and its three pluggable transports using Selenium.

CAIDA background traffic: We use CAIDA's 2018 anonymized traces[6] as a dataset for additional background traffic. We extracted the flows in this database based on the protocol type, IP addresses and port numbers of the end-hosts. For each IP address, we consider all traffic to and from that IP as the typical traffic of that user. Table 2 shows the class breakdown of CAIDA dataset used in our experiments.

HTTP traffic: We collect top 500 Alexa websites using Selenium Tool. Also, we capture these websites over Tor, and three pluggable transports. Moreover, we use the dataset by [**?**] which has collected the top 50, 000 Alexa websites over Tor.

Table 2: Traffic Class Breakdown For CAIDA Dataset

Traffic Class	**Port Numbers**	**♯ of Connections**	**%of Total**
http, https	$80, 8080, 443$	745262	0.318
dns	53	1073758	0.457
smtp	25	2646	0.001
telnet	23	6958	0.003
ssh/scp	22	4928	0.002
other	–	511700	0.219
all		2345252	1.0

5.2 Metrics

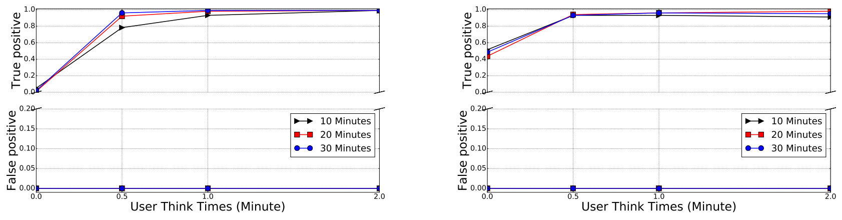
We use following metrics to measure the performance of our classifiers:

- **True Positive:** True positive shows the proportion of the data which contain Bitcoin, and our model correctly identified as Bitcoin traffic.
- **False Positive:** False positive shows the proportion of the data which did not contain Bitcoin traffic, and our model incorrectly classified as Bitcoin.
- **Accuracy:** Accuracy shows the proportion of the data which was correctly classified.

5.3 Modeling Normal Users

In this section, we describe four different types of users' profile that we use to evaluate the performance of our Bitcoin classifiers against. Users may have some background traffic while using Bitcoin or they can generate noisy traffic to evade

[5] `http://www.seleniumhq.org`
[6] `https://www.caida.org/data/monitors/passive-equinix-nyc.xml`



Compact Mode Full Mode

Fig. 14: Result of `sizeHist` classifier on noisy Bitcoin traffic

detection. Therefore, we attempt to model typical behavior of users in various scenarios to evaluate the performance of our system.

– Simple User: A simple user is a Bitcoin client with no background noise and traffic. This type of user does not generate any network traffic except the Bitcoin client application traffic. Therefore, all traffic of the simple user is Bitcoin traffic.
– Simple Noisy User: A simple noise user is a Bitcoin client who browses only **one** webpage.To control the background traffic, we introduce a parameter named think time, T, representing the amount of time that the user spends on a particular website. Note that, increasing T would decrease the background noise since there is not much traffic after a website is loaded. Thus, if T is large and the Bitcoin application is running, after the webpage is completely loaded, the simple noisy user's traffic would look like the simple user's traffic.
– Complex Web (Complicated/Sophisticated) User: A complex user is a Bitcoin client who browses **multiple** websites simultaneously. The complex web user is the sophisticated version of the simple noisy user. To create a sample data of this user profile, we choose a Bitcoin traffic with length of sample size and accumulate the noise traffic using following algorithm:
  1. Choose a random number (k) in the range [0, *sample size*], which represents the length of noise flow.
  2. Choose another random number (p) in the range [0, *sample size*$-k$], which shows where we need to add the noise: p's second of the Bitcoin traffic.
  3. Repeat 1.

  We repeat this process for I number of times, which represents the number of open tabs. The reason that we do not add noise from start to the end of the flow is that we want to make the background noise nonuniform, thus prevent the classifier from learning the noise and denoising the traffic.
– Complex CAIDA User: A complex CAIDA user is a Bitcoin client who is running 1-5 number of CAIDA applications, which is introduced in table 2 simultaneously in the background. We use the same algorithm as above to create this user profile.

We define four different user profiles from simple behavior to more realistic and complicated ones. We apply each classifier on one or more number of these profiles to evaluate their performance.

6 Results

In this section, we implement our classifiers to evaluate their performance on user profiles described in part 5.3 writing more than a thousands lines of code in Python. First, for each classifier, we declare the user profile(s) that we use for evaluation of its performance and the false data that we use for computing its false positive. Second, we describe the result of each classifier and give a summary and comparison of them at the end of this section.

6.1 User Profiles and False Data

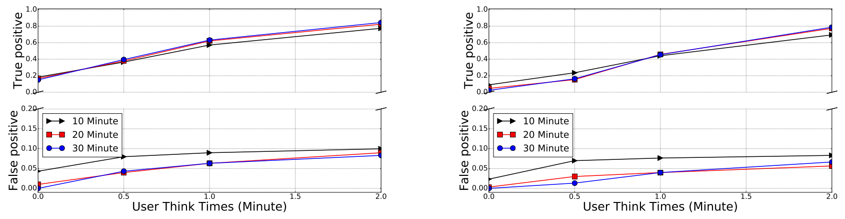
For each classifier, we use a specific user profile and depending on that we choose the false data. As we explained above, false data is the base traffic that we use to compute false positive. In other words, it is the traffic that we compare our Bitcoin traffic with. In the following, we describe these pairs for each classifier(s).

– For window-based classifier, we use the simple user profile. Furthermore, we use HTTP which is the typical user behavior as the false data. This experiment evaluates if Bitcoin traffic can be differentiated from browsing an HTTP website.
– For the rest of the binary classifiers in this section, we use simple noisy user profile and attempt to detect the presence of Bitcoin. Note that, similar to the window-based classifier, we use HTTP for the false data. This experiment attempts to evaluate if browsing an HTTP website is enough to hide the Bitcoin traffic.
– For the neural network-based classifier, we use the complex web and complex CAIDA user for training and testing.

Note that, for the neural network classifier, we evaluate our model using $10,000$ number of test data. For rest of the classifiers, we use 500 number of test data for evaluation.

6.2 Size-based Classifiers

`SizeHist` **Classifier** For this classifier, we compute the histogram of packet sizes and using a correlation algorithm we decide if the traffic contains Bitcoin or not. Because of the Bitcoin specific packet sizes, we expect to have a good performance even in the presence of background noise. We implement the `sizeHist` classifier on Bitcoin traffic in compact and full block modes using the noisy user model described in Section 5.3. Figure 14 shows the impact of traffic length and background noise on true positive and false positive for this classifier. As we explained before, we control the noise using T. As it is expected, increasing T and therefore, decreasing the noise enhances the classifier's performance. Figure 14



Full Mode Compact Mode

Fig. 16: Result of `D2U` classifier on noisy Bitcoin traffic

shows that we can reach more than 90% true positive and 0% false positive for both modes when we have 10 minutes of traffic and set T to 2 minutes. It is worth stating that we could reach similar results when we set T to 0.5 minutes and have 20 minutes of traffic.

The `D2U` Classifier We showed in Section 3 that downstream to upstream ratio of Bitcoin traffic could be a distinguishing factor to identify Bitcoin from other traffics. D2U classifier attempts to use the symmetry between upstream and downstream of Bitcoin traffic to distinguish it from other protocols. Figure 16 shows the result of this classifier on noisy user profile for full and compact block modes. It indicates that increasing T and thus decreasing the background noise on Bitcoin traffic would improve the detection rate. More specifically, our true positive enhances from 0 to 80% when we increase T from 0 to 2 minutes.

6.3 Shape-based Classifier

As we explained in Section 4.2, window-based classifier attempts to detect Bitcoin blocks using the volume of traffic downloaded at a time window around the block announcements times, and using the block detection rate it computes the true and false positive. To implement the window-based classifier, we set J and ω introduced in Section 4.2 to 100 kilobytes and 20 seconds, respectively. To set η, we compute block detection rate for Bitcoin using ground false shown in Figure 17. As we discussed in Section 4.2, we need to set η to be larger than all the detection rate values for Bitcoin using ground false. As we explained in Section 4.2, η is the threshold that we use to differentiate Bitcoin traffic: if the block detection rate is higher than η, we classify the traffic as Bitcoin. Note that each point for Bitcoin using ground false in the figure is the average for 25 different ground false. Using this figure, we set η to 0.4. Moreover, Figure 17 shows the block detection rate for HTTP using ground truth and block detection rate for Bitcoin using ground truth too. Using the η, we can reach detect all Bitcoin traffic through (August 28 - Oct 5) as Bitcoin. Also, we did not classify any of the HTTP traffic as Bitcoin, which results in 0% false positive. Furthermore, the performance of window-based classifier quickly diminishes in the presence of a small HTTP background noise since HTTP noise dominates the Bitcoin traffic and destroys the results of the classifier. Furthermore, we implement the window-based classifier on Bitcoin compact mode and learn that this classifier fails to detect Bitcoin traffic in this mode because of small block sizes, which makes it impossible to distinguish them.

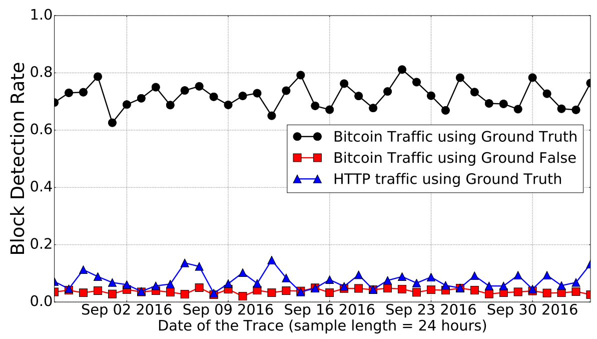


Fig. 17: Block detection rate using window-based classifier

6.4 Neural Network-based Classifier

In this section, we implement the NN-based classifier using Keras [?] with Tensorflow [?] backend. We use complex web and complex CAIDA user to evaluate NN-based classifier. To run this classifier, we need to set two parameters: learning rate and epochs. Learning rate is a hyper-parameter which controls who much we adjust the weights of the neural network model in each iteration. Moreover, epochs depict the number of times that the algorithm is run on the training data. We use the default value of 0.01 for the learning rate of Adam optimizer (lr). For the epochs, we run our model for values ranging from 50 to 2000 and realize that using 1000 number of epochs we can get a good performance from our classifier. Having a small value for epoch keeps the model from learning the dataset. On the other hand, having a very large number for it may cause over-fitting. Therefore, we need to pick this value very carefully. Furthermore, increasing the number of epochs would increase the training time. For example, the training time increases from 20 seconds to around 4 minutes when we increase the number of epochs from 50 to 1000 for 5000 number of training data. Therefore, we need to take this into account when the size of training data increases.

Table 3 shows the result of NN-based classifier for different sizes of training data. As the table indicates, increasing the size of training data improves the results of this classifier. The accuracy of the classifier improves from 62% to 97% when we increase the size of training data from 1000 to $20,000$. Note that, true and false positive improves from 44% to 96% and 20% to 2% respectively when we increase the size of training data.

Table 3: Result of Neural Network classifier

Training Size	False Positive (%)	True Positive (%)	Accuracy (%)
1000	20	44	62
5000	11	80	85
$10,000$	6	84	89
$20,000$	2	96	97

6.5 Summary and Comparison of the Results

In the following, we give a summary of results for size-based, shape-based and NN-based classifiers and then, compare their performance.

- **Shape-based Classifier:** In this category, we have the window-based classifier, which results in 100% true positive and 0% false positive for full block mode when there is no background noise, but it fails to detect Bitcoin traffic on compact block mode because of the small block sizes. The performance of this classifier quickly diminishes in the presence of small noise such as simple noisy user model described in section 5.3.
- **Size-based Classifiers:** In this category, we have SizeHist and D2U classifiers.
  - ☐ **SizeHist Classifier:** Using 10 minutes of traffic with a think time of 1 minute, we can reach 0% false positive and more than 90% true positive in both compact and full block modes. Note that it gets a similar result for both cases when we have more than 20 minutes of traffic with a think time of 30 seconds.
  - ☐ **D2U Classifier:** This classifier reaches around 80% true positive and up to 5% false positive for both compact and full modes when there are at least 10 minutes of traffic and the think time is 2 minutes.

  The previous classifiers including D2U, SizeHist and window-based work just when there is a very small background noise or no noise (think time of 2 minutes). Therefore, they are not useful when Bitcoin traffic has a large amount of background noise. To distinguish Bitcoin traffic in the presence of larger noises, we employ NN-based classifier.
- **NN-based Classifier:** To evaluate NN-based classifier, we use Complex web and complex CAIDA users in compact and full block modes. Our NN-based classifier is able to reach near perfect accuracy (97%) with less than 4% false positive when we increase the size of training data to $20,000$. This classifier outperforms all previous ones since it is able to detect Bitcoin traffic using Complex web and complex CAIDA user explained in 5.3.

7 Countermeasures

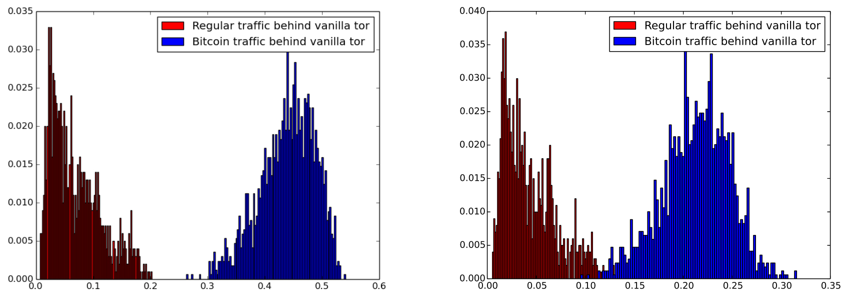
We tunnel Bitcoin traffic over Tor to disguise its patterns. Moreover, we evaluate the invisibility that Tor provides by designing a new classifier (SizeTor) which is tailored for Tor. Also, we use our strongest classifier (NN-based) to evaluate if Tor succeeds in hiding Bitcoin traffic.

7.1 Bitcoin Over Tor

One possible countermeasure against Bitcoin detection is to tunnel Bitcoin traffic over an anonymity system like Tor. Tor sends traffic in *cells*. If the packets sizes is below the cell size, it will add padding to the packets to reach the fixed cell sizes. This modifies the traffic patterns of Bitcoin traffic, e.g., its packet sizes, therefore it may increase resistance to traffic classification. In the following we evaluate our classifiers against Tor and three state-of-the-art Tor pluggable transports [**?**] namely *obfs4*, *FTE*, and *Meek* explained in the following.

Pluggable Transports

- **Obfs4:** obfs4 is a widely used Tor pluggable transport, which is based on *Scramblesuit* [**?**]. It differs with ScrambleSuit in public key obfuscation and its protocol for one-way authentication, which makes it faster.
- **FTE** The FTE transport re-encrypts Tor packets in order to match the regular expressions of a benign protocol like HTTP.
- **meek** Meek uses *domain fronting* [**?**] to tunnel traffic through public CDN or cloud platforms.



upstream downstream

Fig. 19: Histogram of one cell packet ratio for HTTP traffic and Bitcoin traffic

As it is shown in Section 3, the distribution of packet sizes differs in Bitcoin and other types of traffic. While Tor modifies the sizes of Bitcoin traffic, we show that this does not remove all traffic patterns that reveal Bitcoin. we compare the distribution of packet sizes in HTTP traffic and Bitcoin traffic behind Tor. Figures 22a to 22d compare the distribution of packet sizes of HTTP and Bitcoin when tunneled over Tor in different modes, showing an identifiable pattern. We particularly see that Bitcoin traffic has a larger ratio of one-cell packets compared to Tor. This is due to a large number of small Bitcoin messages (e.g., `inv`) that are each put into a single-cell packets. Figures 18a and 18b show the histogram of one cell size packets ratio in the upstream and downstream direction.

7.2 Evaluating Bitcoin over Tor

`SizeTor` Classifier We implement `SizeTor` classifier on noisy Bitcoin on compact mode for Tor and its three pluggable transports. As it is displayed in Figure 20, we could detect Bitcoin traffic with high accuracy for the complex User-1 model when the background noise is one open tab. As Figure 20 indicates, having a traffic size of 10 minutes is enough to detect Bitcoin traffic with around 90% true positive and 0% of false positive. Moreover, the Figure states that the result of classifier quickly diminishes when we increase the background noise from one to 2 or 3 open tabs. Note that this figure shows the average result of this classifier on Tor and three pluggable transports.

Neural Network-based Classifier Table 4 presents the result of NN-based classifier for the complex web user over Tor for $10,000$ numbers of test data. As the table suggests, performance of our classifier improves when we increase the size of training data. More specifically, our false positives enhances from 11% to 4% when we increase the training data from 1000 to $20,000$. Moreover, when using $20,000$ training data, we reach 99% and 4% true and false positive, respectively (97% accuracy). Note that the reason we get better results from this classifier on Tor dataset is that this dataset only contains the complex web user since we did not have CAIDA application over Tor to use for our evaluation.

Table 4: Result of Neural Network classifier for Tor dataset

Training Size	False Positive (%)	True Positive (%)	Accuracy (%)
1000	11	98	93
5000	6	98	96
$10,000$	3	98	97
$20,000$	4	99	97

8 Related Work

In this section, we overview previous work on classifying different protocols and discuss previous attacks on Bitcoin cryptocurrency.

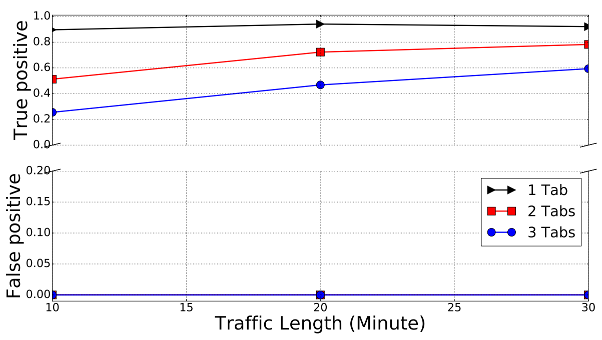


Fig. 20: Detecting Bitcoin compact mode traffic behind Tor (Tor and its three pluggable transports: meek, obfs, fte) Using SizeTor

8.1 Protocol Classification

There is a huge work in literature trying to classify different applications/protocols in the network. Previously, researchers were focused on classifying the applications according to port numbers [?,?,?,?] and payload [?,?,?,?]. There were some problems in the previous methods which made them change their approach. For example, many applications use uncertain port numbers [?], or some encrypt their payloads which makes it impossible to classify them according to payload and port numbers. Recent techniques use statistics such as packet sizes and timings for classification. For example, in [?], authors use a three-level classification mechanism -social, functional and application level- to classify web, p2p, chat, media, gaming, data transfer, streaming and network management traffic. As the authors mention, they do not use any information about port numbers or payload content for their operation. Their result shows that their method can classify $80-90\%$ of traffic flows with more than 95% accuracy. In [?], authors use packet sizes, inter-arrival times and their orderings to distinguish between HTTP, SMTP, SSH, etc. The reason that they choose these features is that at least during the beginning stage of each application these quantities highly depend on the application. They use *anomaly score* to check if a flow is statistically compatible with one of the traffic classes. Note that they compare their result with previous payload-based techniques and state that their approach reaches more than 90% hit-ratio for all the considered classes and false positive around 6% in the worst scenario. In [?], authors use duration, number of bytes, and number of packets to classify flows at the network core where only one side of the flow is available (server-to-client or client-to-server). They propose a method based on syntax and semantics of TCP protocol to estimate the missing statistics, which is due to not having both sides of the flow. They use K-Means algorithm for clustering traffics including Web, P2P and FTP, and use byte and flow accuracy to evaluate their mechanism. They show that server-to-client datasets give the best performance: 95% and 79% flow and byte accuracy, respectively.

In some of the studies, researchers apply machine learning techniques on these features to classify different applications. For example, in [?], authors take a semi-supervised approach to classify a variety of applications such as FTP, HTTP, P2P. They use some statistics such as the total number of packets, average packet size, and total bytes to classify different applications. They show that their technique could reach high accuracy by using a few labeled data and a large number of unlabeled samples. In [?], authors use SVM to classify different applications such as WWW, Mail, and FTP. They show that their approach can reach a good accuracy if they choose the classifier's parameters cautiously. Also, in [?] authors use SVM to classify a broad application category such as Mail, buck traffic, service. They extract statistics such as packet length, byte ratio of sent and received packets, number of packets per flow, window size and transport protocol type as their feature, and show that their method reaches more than 95% accuracy using these features. In [?], Andrew W. Moore et al. use a Bayesian technique to classify different traffic such as P2P and WWW. They show that the simplest Bayesian technique would not provide a good accuracy (65%) and to improve their technique they use two main refinements which would improve the accuracy to 95%. Moreover, in [?], authors use Bayesian neural network to classify Internet traffic using just header-derived statistics. They show that their method is comparable to the ones that use payload or other sophisticated traffic processing techniques. In [?], Wei Li et al. use C4.5 [?] decision tree to classify Internet traffic such as mail, gaming, database, and browsing. They reach an accuracy of 99.8 using 12 features collected at the start of the flow. In [?,?], authors survey the papers on Internet traffic classification using machine learning techniques. To the best of our knowledge, we are the first ones who attempt to distinguish Bitcoin traffic from other applications. We design several binary classifiers to detect Bitcoin traffic in the presence of small background noises. Furthermore, we utilize Neural Networks to detect Bitcoin traffic in the presence of more complex background noises such as browsing more than one website (up to 5) or running applications from CAIDA (up to 5 number of different applications).

8.2 Attacks on Bitcoin Cryptocurrency

In this section, we discuss the previous attacks on the Bitcoin network. In [?], the authors discuss routing attacks, their impact, and possible countermeasures. They study partitioning and delay attacks by investigating node-level and network-wide attacks for both. They adopt delay attack to slow down the propagation of blocks toward a set of nodes. Moreover, they use partitioning attack to isolate a set of nodes in the Bitcoin network by diverting or cutting all the connections from those nodes to the network.

In [?], authors design eclipse attack in which the attacker isolate a victim from its peers by monopolizing its all incoming and outgoing connections. Doing so, the attacker can filter the victim's view of the network and therefore, force the victim to waste her computing power on an outdated view of the network. Furthermore, there have been many deanonymization attacks on Bitcoin.

In [?,?,?,?,?], authors use Bitcoin transaction patterns to link users (or link transactions) using some side information. For example, in [?], the authors design some heuristics to utilize the transaction graph to link transactions to the addresses and eventually infer the identities linked to them.

In [?], the authors propose a new technique to link the public key of a user to her address or link her transactions. They show that they could launch their attacks even when the clients are behind NAT using only a few numbers of machines.

In [?], they analyze the security of using Bitcoin for fast payments and show that the current Bitcoin system is not secure unless they integrate Bitcoin network with some detection mechanism. Also, they study double spending attacks on fast payments and implement a method to prevent it. In [?], the authors introduce selfish-mine in which a pool can obtain a revenue larger than its share of mining power. In this attack, selfish miners hide their mined blocks to create a private branch and selectively reveal them. Doing so, they force the honest miners to move to the new head of the blocks and waste their computing power on the wrong block. In [?], the authors propose a new attack on the Bitcoin payment system that exploits some authentication vulnerability, or some weakness on the refund procedure. They suggest a revision on the Bitcoin payment protocol, which prevents both attacks.

In our paper, we tackle the privacy of Bitcoin users by detecting their Bitcoin traffic even when a user tries to hide its Bitcoin traffic using cover traffic or tunneling through Tor.

9 Conclusions

The reliable access to Bitcoin and similar cryptocurrencies is of crucial importance due to their consumers and the related industry. In this paper, we investigated the resilience of Bitcoin to blocking by a powerful network entity such as an ISP or a government. By characterizing Bitcoin's communication patterns, we designed various classifiers that could distinguish (and therefore block) Bitcoin traffic even if it is tunneled over an encrypted channel like VPN or Tor, and even when it is mixed with background traffic. Through extensive experiments on network traffic, we demonstrated that our classifiers could reliably identify Bitcoin traffic despite using obfuscation protocols like Tor pluggable transports that modify traffic patterns.

We learn from our experiments that it is extremely hard to hide Bitcoin traffic using standard obfuscation mechanism due to specific protocol messages with unique sizes and frequencies. In order to disguise such patterns, an obfuscating protocol needs to apply significant cover traffic or employ large perturbations, which is undesirable for typical clients. We suggest that future work should look into designing obfuscation protocols that are tailored to Bitcoin (and similar cryptocurrencies) in a way to optimize resilience to detection and resource efficiency.

A Bitcoin Protocol Messages

Table 5 lists core Bitcoin protocol messages, which are described in the following.

Table 5: The list of Bitcoin communication messages

Message	Description
`version`	Advertise the node's version. No further communication is possible until both peers have exchanged their version.
`verack`	Reply to the `version` message.
`addr`	Send information about the known nodes of the network.
`inv`	Sent to advertise the knowledge of the peer about the known objects. It can be received unsolicited, or in reply to `getblocks`.
`getdata`	Sent in response to the `inv` message to retrieve information about the content of an object.
`notfound`	If the receiver of `getdata` cannot return the requested information, it respond with `notfound`
`getblocks`	It return an `inv` message with the list of block after the specified block in `getblocks` request
`getheaders`	It return a `headers` message with the list of block after the specified block in `getblocks` request
`tx`	Sent to describes a Bitcoin transaction in response to a `getdata` message.
`block`	`block` message is sent in response to a `getdata` message
`headers`	Return a list of block headers, in respond to `getheaders`
`getaddr`	A node sends `getaddr` to ask about the known peer from other peers
`mempool`	It asks about the transaction in mempool of other peers
`ping`	Show the TCP/IP connection is still valid.
`pong`	Response to `ping` message.
`reject`	It show a message has been rejected
`sendheaders`	let other peers to send headers without `inv` message
`sendcmpct`	let other peers to send compact blocks
`cmpctblock`	it used in stead of `block`, to send cmpctblock
`getblocktxn`	it indicate missing block in compact block transaction
`blocktxn`	To send missing block in compact block transaction

A.1 Synchronization Messages

These messages are aimed at keeping Bitcoin peers synchronized with the rest of the Bitcoin network.

`addr:`Each peer advertises the information and IP addresses of other peers via `addr` message in the network. `addr` message contains count and list of other peers IP addresses. Each IP address is accompanied by a timestamp showing its freshness. When a peer receives a list of addresses from other peers, it has a choice to forward any number of them. The peer chooses the sending addresses based on the following criteria: 1) The number of IP addresses in the received message should not be greater than 10,and 2) The timestamps should not be older than 10 minutes. This mechanism is applied for helping in peer discovery.

`inventory(inv):` Peers send `inv` to advertise their knowledge about the known objects, like transactions and blocks. Each `inv` message consist of number of inventory entries and the inventory vectors itself. It can be received unsolicited, or in reply to `getblocks`. Inventory vectors are used for notifying other nodes about objects they have or data which is being requested. Inventory vectors consist of the type of objects and the hash of the object.

`getdata:` A peer sends `getdata` message in response to the `inv` to retrieve information about the content of an object. This object could be a block or a transaction.

`tx:` This message describes a bitcoin transaction in response to a `getdata` message. Each transaction is stored in a memory pool. If a received transaction is already in the pool, or it is included in one of the blocks in the main block-chain, it get discarded.

A.2 Block-Related Messages

Such messages are used to exchange Bitcoin blocks among the peers. The current Bitcoin network is supporting two ways of propagating blocks, full block and compact block propagation. Figure 21 demonstrates the flow of message communication in these two ways.

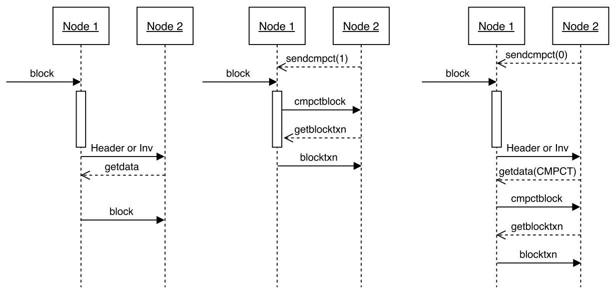


Fig. 21: Different relaying modes in Bitcoin

Full Block Propagation: The sender node first validate the block completely, then it advertise the possession of block by `inv` message. The receiving peer which doesn't have the block, asks for it by sending `getdata` message. Finally, the sender node send the block via `block` message. Sending full block in the network is wasting network bandwidth since we are re-sending all of the transaction and nodes have some of transaction in their memory pool. The messages transmitted in this mode are:

`block:` It consist of block version information, previous block hash, merkle root of a Merkle tree collection which is a hash of all transactions related to this block. Sending the new block forwarded through all the network.

Compact Block Propagation: From the middle of 2016 in `0.13.0` version, Bitcoin protocol start to forward blocks as compact blocks which means instead of forwarding full blocks in network, only a sketch of block is sent. The sketch include 80-byte block's header, the short transactions IDs used for matching already-available transactions and select of transactions which sending peer expect that a receiving peer may be missing. After receiving the compact block, the receiving peer tries to reconstruct the block at its end, from the already received transactions and the one in compact block. In this way the waste of sending each transaction twice is reduced. The advantage of compact block relaying is reducing the spikes in the bandwidth and also reduce propagation delay. The messages transmitted in this mode are:

`sendcmpct:` This message is introduced in the compact block relaying and it is used to inform the receiving peer about the mode of communication the sending peer has chosen. If the first byte of the message is set to 1 the sender is indicating that it wants to receive blocks as soon as possible and it is working in the high-bandwidth mode. If the first byte of the message is set 0 the sender is saying that it wants to minimize bandwidth usage as much as possible and it is working in the low-bandwidth mode.

`cmpctblock:` This message introduced in the compact block relaying and is presenting a sketch of block.

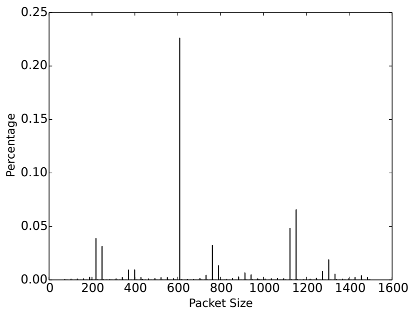
`getblocktxn:` This message is introduced in compact block relaying and is used to request for the transactions that are missed by sending a list of their indexes.

`blocktxn:` This message is introduced in compact-block relaying and is used to provide some of the transactions in a block, as requested.

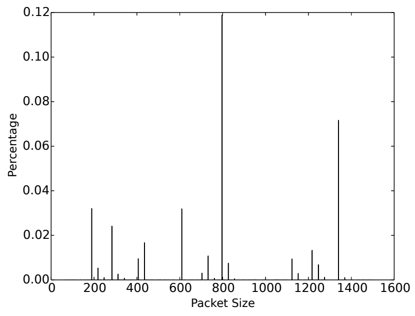
Compact block relaying works in high and low bandwidth settings. In high bandwidth the receiving peer doesn't oblige the sending peers to ask for permission first. So, multiple peers can send the compact block to receiving node. Then at last the sender node sends the missing transactions by `blocktxn` message. It is worth mentioning Bitcoin works in high bandwidth mode with up to 3 peers. However, in low bandwidth mode, since bandwidth is its bottleneck, the receiving node oblige other nodes to ask for permission first. So, the sender first advertise the block possession by `inv` message. Then, the receiving node asks for the compact block by `getdata` and the sender will send the compact block by `cmpctblock` message. At last, if there is any missing transaction, the receiving node will ask for it by `getblocktxn` and the sender will send those transactions via `blocktxn`.

B Bitcoin Traffic Shape

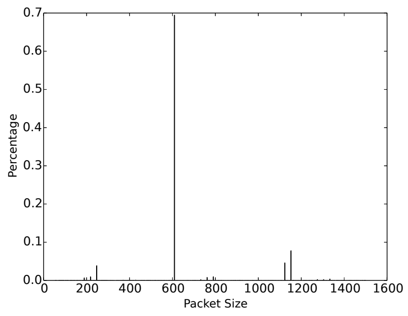
Figure 22a-22f shows the upstream and downstream packet distribution of HTTP and Bitcoin traffic behind Tor.



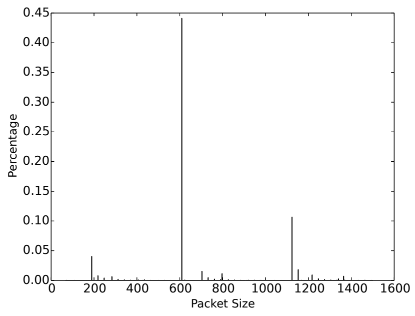
HTTP, upstream



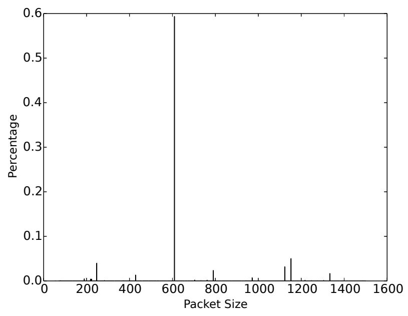
HTTP, downstream



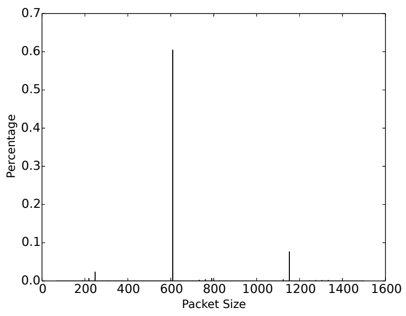
Bitcoin, Compact block, upstream



Bitcoin, Compact block, downstream



Bitcoin, Full block, upstream



Bitcoin, Full block, downstream

Fig. 23: Distribution of packet sizes of HTTP and Bitcoin traffic behind Tor