TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 13/15 pp702-713
Volume 22, Number 6, December 2017

An Active De-anonymizing Attack Against Tor Web Traffic

Ming Yang*, Xiaodan Gu, Zhen Ling, Changxin Yin, and Junzhou Luo

Abstract: Tor is pervasively used to conceal target websites that users are visiting. A de-anonymization technique against Tor, referred to as website fingerprinting attack, aims to infer the websites accessed by Tor clients by passively analyzing the patterns of encrypted traffic at the Tor client side. However, HTTP pipeline and Tor circuit multiplexing techniques can affect the accuracy of the attack by mixing the traffic that carries web objects in a single TCP connection. In this paper, we propose a novel active website fingerprinting attack by identifying and delaying the HTTP requests at the first hop Tor node. Then, we can separate the traffic that carries distinct web objects to derive a more distinguishable traffic pattern. To fulfill this goal, two algorithms based on statistical analysis and objective function optimization are proposed to construct a general packet delay scheme. We evaluate our active attack against Tor in empirical experiments and obtain the highest accuracy of 98.64%, compared with 85.95% of passive attack. We also perform experiments in the open-world scenario. When the parameter k of k-NN classifier is set to 5, then we can obtain a true positive rate of 90.96% with a false positive rate of 3.9%.

Key words: traffic analysis; active website fingerprinting; anonymous communication; Tor

1 Introduction

As people become increasingly aware of network security and privacy concerns, significant efforts have been exerted to develop anonymization techniques to protect the data privacy of users[1], as well as the privacy of their locations[2] and communications. For the latter, various anonymous communication systems have been proposed and widely used, e.g., OpenSSH, JAP, Tor, and I2P. Owing to the high security performance, the popularity of Tor has grown throughout the world. According to official statistics[3], by the end of March 2017, more than 2.38 million users access the Internet through Tor, including 180 000 clients that connect via bridges. To obscure the communication relationship, Tor always selects three nodes (entry, middle, and exit nodes) to build a path. Furthermore, Tor packs all data into 512-byte transmission units, which are called cells, to make them indistinguishable.

To compromise the anonymity provided by Tor, traffic analysis techniques are studied[4–8]. They can be categorized into two groups[9], namely, end-to-end attacks[7, 8] and single-end attacks[4–6]. In the end-to-end attacks, an adversary should first control both the entry and exit node of a path used by a Tor client. Then she can passively observe the traffic patterns on both sides to correlate the traffic of the sender and receiver. Alternatively, she can actively modulate the traffic to embed a watermark on one side and then inspect the watermark[7, 8] on another side to confirm the communication relationship between the sender and the receiver. In single-end attacks, also referred to as Website Fingerprinting (WFP) attacks, an adversary can passively observe the patterns of the traffic between the Tor client and the entry Tor node to infer the websites visited by Tor users. In practice, such type of adversaries can be an ISP or a local network administrator. Therefore, it can post a significant threat to the security and privacy of Tor

• Ming Yang, Xiaodan Gu, Zhen Ling, Changxin Yin, and Junzhou Luo are with School of Computer Science and Engineering, Southeast University, Nanjing 211189, China. E-mail: yangming2002@seu.edu.cn; guxiaodan@seu.edu.cn; zhenling@seu.edu.cn; yinchangxin@seu.edu.cn; jluo@seu.edu.cn.
* To whom correspondence should be addressed.
Manuscript received: 2017-07-17; accepted: 2017-07-26

users with the least resources because the adversary in website fingerprinting is one of the weakest adversaries in the threat model of existing attacks.

An intense research effort has been made to demonstrate the feasibility of the website fingerprinting attack against Tor in recent years[6, 10–12]. The website fingerprinting attack is always converted to a classification problem. In general, these studies improve the attack by exploring new features (such as the bursts and concentrations of outgoing packets[11]) and modifying the classical classification algorithms. However, all these studies are limited to the passive threat model and the extracted features are sensitive to the network dynamic. For example, we accessed the CNN website through the same Tor circuit twice in three minutes, and found that both burst packet numbers and burst volumes varied dramatically and can affect the accuracy of the website identification. To address the problem, He et al.[13] proposed an active website fingerprinting attack. The main idea is to identify and delay the HTTP requests from the Tor client to separate the mixed traffic that carries various web objects. However, since all the traffic is encrypted, it is nontrivial to identify the right HTTP requests. Thus, it spends more time inferring each HTTP request and leads to a low accuracy of website inference.

In this paper, we propose a novel active website fingerprinting attack. The main contributions of our work are as follows.

First, we propose a new threat model for the active website fingerprinting attack against Tor. In this threat model, an attacker can control an entry node and manipulate appropriate cells transmitted between the entry and middle nodes.

Second, we design two algorithms based on statistical analysis and objective function optimization, respectively, and build a general cell manipulation framework to delay five proper cell positions from a sequence of cells at the entry node so that the accuracy of the website identification can be improved.

Third, we design a delay scheduling algorithm for all the selected cell positions to separate the response traffic of different web objects. Thus, we can extract more distinguishable and stable features for website identification. By carrying out experiments in the closed-world and open-world scenarios, we demonstrate the effectiveness and feasibility of our attack.

The rest of this paper is organized as follows. In Section 2, we present the related work. In Section 3, we introduce the background about Tor. We describe the threat model and propose our active website fingerprinting attack in Section 4. We provide the results of our experiments in Section 5. In Section 6, we conclude this paper and discuss directions for future work.

2 Related Work

In the past decades, researchers focused on the website fingerprinting attack against the single-hop anonymous communication systems and extracted the web object sizes, packet sizes, packet ordering, inter-arrival times, and other features to generate website fingerprints. However, these features are disturbed because the data are encapsulated into equal-length transmission units. Herrmann et al.[4] were the first to launch the website fingerprinting attack against Tor. They extracted the packet size distribution and applied multinomial naive Bayes classifier to evaluate the similarity. By using a dataset containing 775 monitored websites, they obtained an accuracy rate of only 3%. In 2011, Panchenko et al.[5] proposed many new features, including total transmitted bytes, percentage of incoming packets, and occurring packet sizes. By applying Support Vector Machine (SVM), they increased the detection rate to 55%. They also introduced a new attack scenario called open-world. They believed that the ability of the attacker is limited and she could not monitor all websites. The attacker first needs to determine whether a website is on the monitoring list or not. If the website is on the list, she also needs to classify it into the right category.

Researchers found that converting packets into cells for feature extraction could improve the accuracy. Cai et al.[6] rounded all packet sizes up to a multiple of 600 and extracted ordering and numbers as features. To improve the performance of the classifier, they used the optimal string alignment distance as the kernel function for SVM. Likewise, Wang and Goldberg[10] reconstructed Transport Layer Security (TLS) records from packets and rounded the resulted lengths to the closest multiple of 512. Furthermore, they designed a heuristic algorithm to remove *Sendme* cells. By using a new distance-based metric in SVM, the accuracy rate reached 91%.

In recent years, website fingerprinting attacks were futher improved by employing new features and

classifiers. Wang et al.[11] extracted six types of features and iteratively adjusted feature weights in the k-Nearest Neighbor (k-NN) classifier. Panchenko et al.[12] sampled features from a cumulative representation, which were robust against dynamics of bandwidth, congestion, and page load time. They also implemented large-scale experiments to evaluate the effectiveness. Hayes and Danezis[14] extended the random forest technique to select appropriate features and conducted experiments over standard web pages as well as Tor hidden services.

Juarez et al.[15] believed that some assumptions of the attacks were not practical and might make a significant impact on the efficacy. To address these issues, Wang and Goldberg[16] discussed several methods to bridge the gap between laboratory and realistic conditions for website fingerprinting.

Overall, the aforementioned passive website fingerprinting attacks cannot deal with the web objects mixed in the traffic. To address this problem, He et al.[13] proposed an active attack by delaying HTTP requests originating from users for a certain period, which separated responding traffic containing various web objects. However, they could not accurately identify the HTTP requests and filter all the control packets when they modulated the traffic between the Onion Proxy (OP) and the entry node. It caused a low accuracy.

3 Background

Tor is a popular low-latency anonymous communication system that supports TCP-based applications. The Tor network consists of three components: the OP, Onion Routers (ORs), and directory servers. The OP selects three ORs by default and establishes circuits hop by hop. Multiple TCP streams can be multiplexed into a single circuit. To achieve high-performance scheduling, Tor uses libevent[17] to schedule the read/write events.

Figure 1a illustrates the structure of an equal-sized Tor cell, including a 3-byte cell header and a 509-byte payload. The cell header is plaintext, while the 509-byte payload of a Tor cell is encrypted. The cell header consists of two fields called Circ_ID (2 bytes) and CMD (1 byte), where the Circ_ID specifies the circuit number and CMD indicates the category of the cell command. Two types of cells, i.e., control cells and relay cells, are used. The CMD of a control cell contains

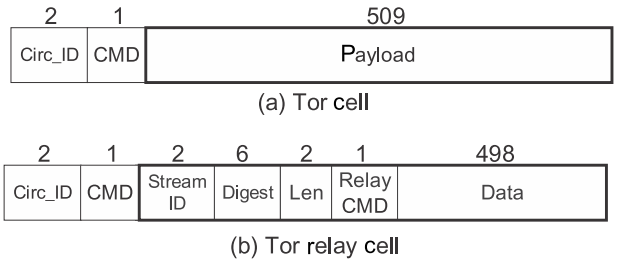

Fig. 1 Tor cell format.

padding, *create*, *created*, *destroy*, and other five values. Moreover, Tor introduced a new type of relay cell in 2008, referred to as *Relay Early* cell, to prevent users from building infinitely long paths[18], and the value of this CMD field was 9[19].

Figure 1b depicts the structure of a relay cell with a CMD value of 3. It has an additional header at the front of the payload, containing the StreamID, Digest, Length, Relay CMD, and Data fields. The Relay CMD further specifies the subcategory of the payload: *relay data*, *relay begin*, *relay end*, *relay sendme*, and so on. The *relay data* cells are used to transmit application data, while the rest relay cells, referred to as relay control cells, are used to carry control command. The 509-byte payload of a Tor cell is encrypted in the onion-like fashion. When the cells are delivered to the TLS layer, the cells are encrypted again. Thus, an adversary cannot identify the categories of cells if she monitors traffic on the wire. However, if an adversary controls the entry node of the circuit used by Tor client, she can observe the plaintext of the 3-byte cell header and identify the type of cell in terms of the CMD field.

Figure 2 illustrates the process of creating a circuit and a stream as well as transmitting data over the stream. TLS tunnels are first established among OP and ORs. Then, to create a circuit, an OP sends a *create* cell to establish the first hop circuit with the OR1, which is referred to as entry node. Then, a *relay extend* cell is encapsulated into a *relay early* cell to require the entry node to extend the circuit to the second hop node, which is called middle node. Upon deriving this *relay extend* cell, the entry node extends the circuit on behalf of OP by sending a *create* cell to the middle node. Likewise, the middle node is asked to extend the circuit to the third hop, which is referred to as exit node. At this point, a 3-hop circuit is completely established. The traffic sent toward the server via the circuit is called outbound traffic, while the traffic sent toward client is called inbound traffic. In addition, to avoid infinite-

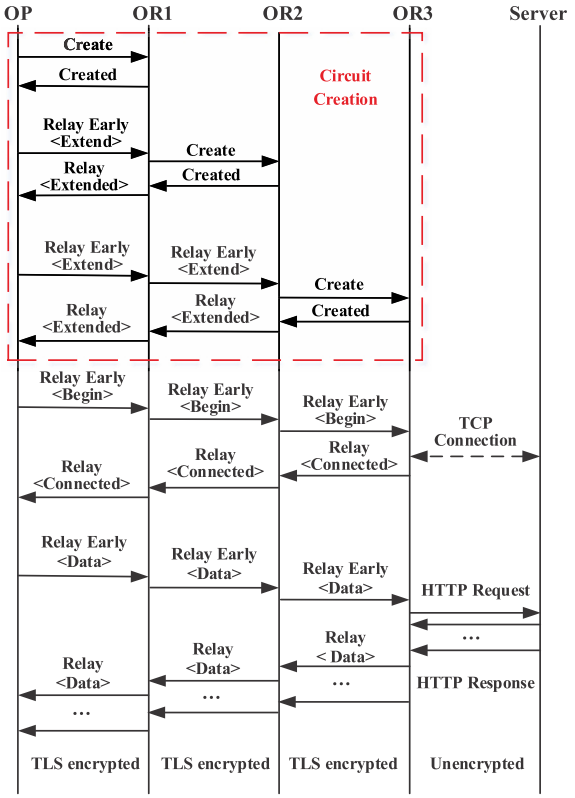

Fig. 2 Circuit creation and data transmission.

length circuit attacks and hide the circuit length, the OP should pack the first 8 data cells in this circuit into *relay early* cells. Therefore, the *relay extend* cell is the first one encapsulated into a *relay early* cell.

After the circuit is established, the user can open an application, e.g., web browser. Then, the OP creates a stream over the circuit for transmitting the data from the web browser. A *begin* cell, including the information of the IP and port of the remote server, is packed into the *relay early* cell to require the exit node to connect to the remote web server. Once the TCP connection between the remote server and exit node is built, the *connected* cell is sent back to the OP to inform it of the connection status. Subsequently, the OP can transmit the data for the web browser over this stream. In fact, the OP multiplexes multiple streams over this single circuit to improve efficiency and anonymity.

4 Active Website Fingerprinting Attack

In this section, we first present the threat model and then introduce our attack, including the cell-delay position decision algorithm and delay scheduling algorithm.

4.1 Threat model

In our threat model, we assume that an adversary controls the entry node in a circuit built by victim's OP. Then she can actively manipulate the traffic at the entry node, i.e., delaying the cells. However, the attacker cannot tamper with the content of the cells. Since Tor nodes are deployed by many volunteers around the world, this assumption is reasonable and practical. To run a stable node in the Tor network, the attacker only needs to meet certain requirements, e.g., Mean Time Between Failures (MTBF) and sufficient bandwidth. Once the attacker's node satisfies the requirements of the MTBF and bandwidth, her node can become the entry node[20]. According to the node selection mechanism, a powerful attacker can deploy enough bandwidth of Tor entry nodes; then, a possiblility is that the OP selects one of the attacker's entry nodes in terms of existing analysis[21–23].

4.2 Basic idea

Figure 3 illustrates the basic idea of active website fingerprinting attack. We manipulate and record the transmitted cells at the entry node and then perform the website fingerprinting attack to infer the website accessed by Tor users. To this end, we first have to understand Tor's communication mechanism and determine when the HTTP data are transmitted. We identify the positions of HTTP requests in the cell sequence from the outbound traffic. Furthermore, we design two algorithms to decide which cell should be delayed and keep all subsequent cells (including the current one) in a delay queue. When a specific delay time is up, all cells in the queue are delivered to the network until they reach the next cell-delay position. Finally, we record the entire modulated cell sequences and extract appropriate features to classify the real

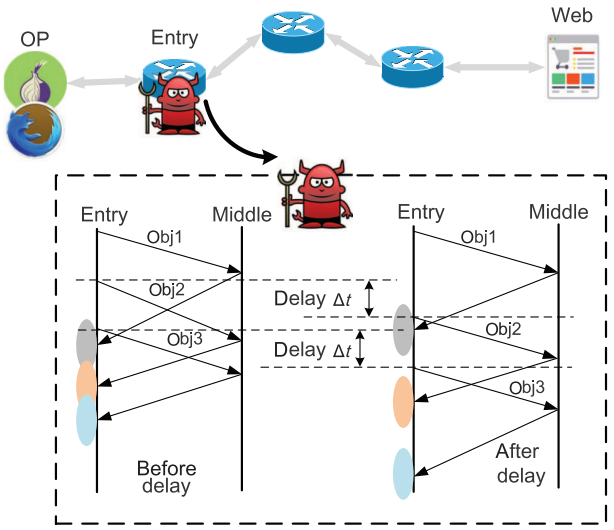

Fig. 3 Basic idea of active website fingerprinting attack.

destination website.

Figure 4 illustrates the workflow of our active website fingerprinting attack. It consists of the following five steps:

- **Step 1:** Determining the first HTTP request. Since the HTTP data encapsulated into Tor cells are transmitted through a three-hop circuit, it is nontrivial to directly identify the HTTP request at the entry node. To address this issue, we first analyze the Tor protocol behaviors, including circuit and stream establishment, as well as data cell transmission. Then, we can accurately infer the procedure of the circuit and stream creation. At this point, the HTTP data can be transmitted between OP and remote web server, and the first outbound cell is the first HTTP request.
- **Step 2:** Delaying HTTP requests. When OP browses a certain website, the entry node may receive many relay cells originating from the OP, including relay data (HTTP requests) and multiple relay control cells. Since all these relay cells are encrypted, directly identifying each HTTP request from the cells is impossible. In addition, the HTTP request positions of each website in the cell sequence are different. Thus, we have to design a general and effective scheme to find several appropriate HTTP request positions and then delay them to separate the web objects. To this end, we design two algorithms based on statistical analysis and objective function optimization, respectively. Furthermore, we modify the code of the Tor transmission mechanism to delay the selected cells.
- **Step 3:** Recording Tor cells. We record Tor *relay* and *relay early* cells at the entry node for feature extraction.
- **Step 4:** Feature extraction. To generate a fingerprint, we extract various features from the recorded cell traces, including total per-direction bandwidth, outbound cells' positions, concentration of inbound cells, and others.
- **Step 5:** Website classification. Based on the extracted features, we apply two classifiers in the closed-world scenario and compare their prediction ability. Finally, we choose a better one and evaluate its practical feasibility in the open-world scenario.

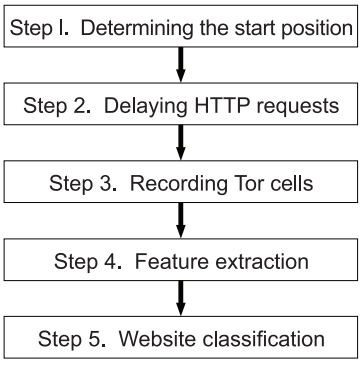

Fig. 4 Workflow of active website fingerprinting attack.

4.3 Step 1: Determining the first HTTP request

To locate the first HTTP request, we determine when the HTTP data are transmitted. As shown in Fig. 2, once a circuit and a stream are built, the OP begins to transmit the HTTP data. Therefore, we can observe the circuit and stream establishment process at the entry node and infer the cell position of the first HTTP request. We record the cell transmission pattern, including the cell sequence, circuit ID, and types of cells.

From these recorded cells, we infer the cell that packs the first HTTP request. Once the OP starts, it creates four circuits by default. After these four circuits are built, users can open the browser to access a website via OP. Then, OP picks one of these circuits and establishes TCP streams over this circuit to transmit HTTP data. The StreamID field in the encrypted cell payload is used to distinguish various TCP connections established by the victim's browser. By using Firefox to browse the websites through Tor, we find that the OP always builds two TCP streams before sending the first HTTP request. At the entry node side, we can obtain a 3-byte cell header in plaintext for each received cell. Since all HTTP data are encapsulated in the relay data cells, we only record two types of cells relayed between the entry and middle nodes, i.e., the *relay* cells and *relay early* cells, whose values in the CMD field are 3 and 9, respectively. Recall that the first 8 *relay* cells should be encapsulated into *relay early* cells by OP. Since the first *extend* cell sent by OP is used to build the second hop circuit as shown in Fig. 2, we can only record 7 *relay early* cells and the rest should be *relay* cells from the outbound traffic at the entry node. Therefore, if a circuit is established and used to transmit HTTP data, we can infer the types of the first 4 *relay early* cells in this circuit. The first *relay early* cell is used to create circuits and subsequent two are *relay begin* cells used to establish two TCP streams. The fourth cell encapsulates the first HTTP request, which is the start position of the HTTP data transmission. Since four circuits are established in advance, three extra *relay*

early cells are used to creat the other three circuits. Thus, the first HTTP request is the seventh cell in the entire cell sequence.

Figure 5 depicts an example of the cell transmission between entry and middle nodes when a Tor user browses a website. We can observe the process of the four circuit creations. Then, the entry node relays two cells in the circuit with circuit ID 2315862583 to open two streams. Therefore, the seventh outbound cell is the first HTTP request.

4.4 Step 2: Delaying HTTP requests

4.4.1 Deciding cell-delay positions

When we find the position of the first HTTP request, we should decide the following delay positions for other requests. In the following data transmission, the OP sends the *relay begin* cell to open a new stream repeatedly. Besides, many *relay sendme* cells are in the outbound traffic. Tor defines two types of *relay sendme* cells, namely, *circuit-level sendme* and *stream-level sendme* cells. Both are used for congestion control. When the OP receives 100 cells per circuit, it sends a *circuit-level sendme* cell. Likewise, it sends a *stream-level sendme* cell after receiving 50 cells. Since all the relay cells are encrypted, we cannot identify the specific positions of HTTP requests from a series of the *relay* cells. In addition, sequences of cells generated by accessing each website may contain different numbers of HTTP requests located in distinct positions. As a result, we should design a general scheme to delay these HTTP requests. To solve these issues, we visit each website on our monitoring list 60 times and analyze the positions of HTTP requests by decrypting the traffic at the OP side.

```
->->->->    1st cell, Circ_id--3035614764, cmd-- 9.
<-<-<-<-    Circ_id--3035614764, cmd-- 3.
1: 3035614764 |
->->->->    2nd cell, Circ_id--3533272601, cmd-- 9.
<-<-<-<-    Circ_id--3533272601, cmd-- 3.
1: 3035614764 | 2: 3533272601 |
->->->->    3rd cell, Circ_id--3767398131, cmd-- 9.
<-<-<-<-    Circ_id--3767398131, cmd-- 3.
1: 3035614764 | 2: 3533272601 | 3: 3767398131 |
->->->->    4th cell, Circ_id--2315862583, cmd-- 9.
<-<-<-<-    Circ_id--2315862583, cmd-- 3.
1: 3035614764 | 2: 3533272601 | 3: 3767398131 | 4: 2315862583
->->->->    5th cell, Circ_id--2315862583, cmd-- 9.
                          → Open a stream
->->->->    6th cell, Circ_id--2315862583, cmd-- 9.
                          → Open a stream
<-<-<-<-    Circ_id--2315862583, cmd-- 3.
->->->->    7th cell, Circ_id--2315862583, cmd-- 9.
                          → The first HTTP request
<-<-<-<-    Circ_id--2315862583, cmd-- 3.
```

Fig. 5 Cell transmission between entry and middle nodes.

We design two algorithms, one based on statistical analysis and the other on objective function optimization. In the first method, we compute the probability that the i-th cell in the outbound cell sequence contains an HTTP request. The results show that the probability of the 7th cell containing an HTTP request is 100%. The 8th cell has a probability of 40% and the 9th has a probability of 20%. For 60% websites, at least 3 HTTP requests are sent in the first 19 cells. Therefore, we decide to delay the $(i+1)$-th, i.e., 8th, 9th, 10th, and 20th cells.

We also design the cell-delay position decision algorithm by optimizing the objective function as shown in Algorithm 1. In this algorithm, we define the concept of conflict in an instance of a monitored website as the absolute difference between the mode and the number of requests before a potential cell-delay position. We compute the number of conflicts in each potential cell-delay position and find corresponding positions in terms of top-T minimum values of the conflicts, where T represents the number of cell-delay positions. For the w-th website, we denote the number of HTTP requests before the p-th cell in the i-th instance as $r_{p,w,i}$ and we can have a set $R_{p,w} = \{r_{p,w,1}, \ldots, r_{p,w,n}\}$. We denote $M(R_{p,w})$ as

Algorithm 1 Cell-Delay Position Decision Algorithm

Input:
(a) P, list of all positions;
(b) m_list, list of monitored websites;
(c) K, total number of monitored websites;
(d) $instances[1, \ldots, K]$, instances collected for all websites in m_list;
(e) n, number of instances collected for each website;
(f) T, number of delay positions to select.

Output: T delay positions.

1: **for** each position p in P **do**
2: **for** each website w in m_list **do**
3: **for** each instance i in $instances[w]$ **do**
4: Calculate $r_{p,w,i}$
5: **end for**
6: Calculate the mode $M(R_{p,w})$ of $R_{p,w}$
7: Calculate the conflicts for w at position p: $S_{p,w} = \sum_{i=1}^{N} |r_{p,w,i} - M(R_{p,w})|$
8: **end for**
9: Count conflicts for all websites at position p: $S_p = \sum_{w=1}^{K} S_{p,w}$
10: **end for**
11: Select T positions corresponding to top-T minimum values in $\{S_p\}$

the mode of HTTP requests in $R_{p,w}$. Then, we can compute the conflicts of the w-th website by

$$S_{p,w} = \sum_{i=1}^{N} |r_{p,w,i} - M(R_{p,w})| \quad (1)$$

Finally, we calculate the sum of $S_{p,w}$ for all K websites as the number of conflicts in the delay position of the p-th cell by

$$S_p = \sum_{w=1}^{K} S_{p,w} \quad (2)$$

To obtain the cell-delay positions of top-T minimum conflicts, we use the objective function of the delay positions decision algorithm by

$$Get\text{-}T\text{-}\mathrm{argmin}_{p \in P}(S_p) \quad (3)$$

Figure 6 shows an example of five instances of the w-th website. The square represents an HTTP request while the circle represents a relay control cell. The x axis indicates the positions of cells in the cell sequence, while the y axis is the instance number. The vertical dash line indicates that we delay the first cell after the line, i.e., the 6th cell. Thus, the conflict of the w-th website is 2 in terms of Eq. (1).

We combine the results from two algorithms and select the proper numbers of cell-delay positions using our empirical experiments presented in Section 5.

4.4.2 Delaying HTTP requests

When the cell-delay positions are determined, we modify the code of the Tor cell transmission mechanism to delay the cells. On the one hand, we have to find the right cells and delay them before writing them to the outgoing buffers. On the other hand, the data read and write events should be scheduled normally.

By analyzing the source code of Tor, we obtain the workflow of Tor data processing as shown in Fig. 7. First, data are received by the network adapter and processed by TCP and TLS stacks. When data are sent to Tor, Tor schedules a read event

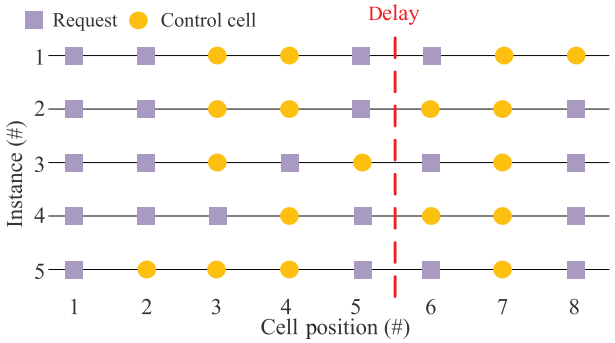

Fig. 6 Example of five instances of website *w*.

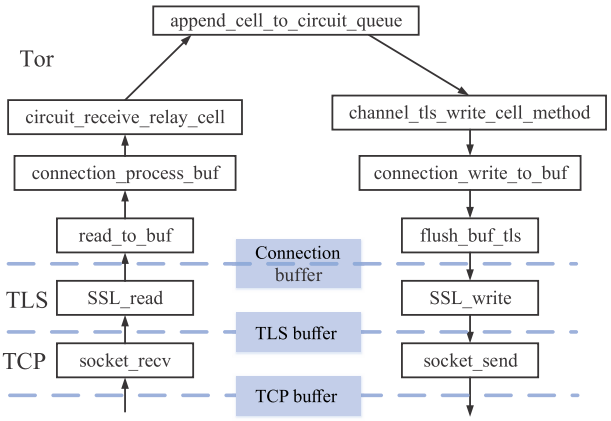

Fig. 7 Tor data processing.

to call the *conn_read_callback* function to load the data to the connection buffer. Then, Tor processes the data in several stages, specifically, sending cells to different modules in terms of the CMD field, decrypting cells in the *circuit_receive_relay_cell* function, and delivering cells to different circuit queues in terms of the circuit number. Once the data are processed, a write event is scheduled to write the data to the connection buffer of the next hop using the *channel_tls_write_packed_cell_method* function. Finally, data are delivered to the *openssl* in chunks and sent to the network through the TCP/IP protocol stack.

To delay HTTP requests, we design a delay scheduling algorithm as shown in Algorithm 2. We count the number of outbound cells and mark the cell if we need to delay it. Then we keep this cell and the following cells in a delay queue. To control the delay time, we set a timer for each marked cell. Once the time is up, we send the corresponding cell and following cells until we count the next delay cell position. Then, the new timer is activated again.

4.5 Step 3: Recording Tor cells

We capture and record all relay cells between the entry node and middle node. To achieve this goal, we capture the inbound and outbound cells in the *circuit_receive_relay_cell* function and *channel_tls_write_packed_cell_method* function, respectively. All captured *relay* and *relay early* cells are recorded in a trace file in chronological order. As all cells are of equal length, we replace data volume with the number of cells in the feature extraction. Then we use the positive and negative signs to represent inbound and outbound cells, respectively.

Algorithm 2 Delay Scheduling Algorithm in Tor

Input: $cell\_queue$, the pending Tor cell queue of $\{cell, seg\}$, seg is true if the Tor cell needs to be delayed.
Output: delayed $cell\_queue$.

```
for every cell in cell_queue do
  if timer_flag is true then
    {The timer is activated now}
    Put the cell into linklist of the delay cell sequence
  else
    if seg of the cell is false then
      Send the cell directly
    else
      Put the cell into linklist
      timer_flag = true
    end if
  end if
end for
Timer function:
if timer is up then
  Send the first cell in linklist directly
  while linklist is not null do
    Get the next cell in linklist
    if seg of the cell is false then
      Send the cell directly
    else
      Start a new timer
      break
    end if
  end while
  if linklist is null then
    timer_flag = false
  end if
end if
```

4.6 Step 4: Feature extraction

On the basis of previous work[11], we remove redundant features and extract various features to generate a fingerprint, including the total bandwidth, per-direction bandwidth and corresponding proportion, outbound cells' positions in the cell sequence, concentrations of inbound cells, and statistical characteristics of bursts. For the outbound cells, we record only the first 300 cells' positions. If the outbound cells of a website are less than 300 cells, we add 0 to fill empty positions. With regard to the bursts, we record the first 50 burst volumes and calculate the maximum, minimum, and mean values. We also count the numbers of bursts whose volumes exceed 5, 10, and 15, respectively.

4.7 Step 5: Website classification

Various factors can affect the classification accuracy of a classifier, such as the size and distribution of a dataset, and the dimension of the features. For the same dataset, distinct classifiers may exhibit varying performance. Therefore, classifier selection is an important step in the active website fingerprinting attack. As one of the most commonly used classifiers, SVMs have good generalization properties and can avoid overfitting on a small dataset. The k-NN algorithm improved by Wang et al.[11] exhibited excellent performance in their experiments. The researchers randomly assigned a value to each weight in initialization, and then iteratively adjusted weights to compute the distance between any two feature vectors.

In this paper, we apply the two classifiers in a closed-world scenario and compare their prediction ability. Finally, we choose the better classifier and evaluate its practical feasibility in an open-world scenario. In the following experiments, we use the LIBSVM library[24] to implement SVM while the k-NN algorithm is provided by Ref. [11].

5 Evaluation

In this section, we deploy a private Tor network in PlanetLab[25] and implement the proposed active website fingerprinting attack to evaluate its effectiveness and efficiency.

5.1 Experimental setup

To set up a private Tor network in PlanetLab, we deploy three directory servers, one bridge, and seven ORs. The experimental setup is shown in Fig. 8. We install the Tor software, version 0.2.5.10 on Fedora release 8 with kernel 2.6.32-20 for all PlanetLab nodes. The OP is installed on Ubuntu 14.04.4 and a script is used to automatically run Mozilla Firefox 47.0 to access the monitored websites. Similar to previous studies, the Firefox cache is switched off. We also disable browser features that may generate noise traffic, such as automatic updates and speculative pre-connections. We use *Obfsproxy*[26] to obfuscate traffic between the OP and bridge. We modify the OP and bridge configuration in the Tor configuration file, as shown in Table 1, to let

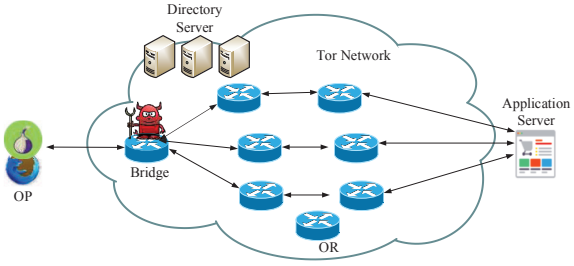

Fig. 8 Experimental setup.

Table 1 OP and bridge configuration file.

OP:
ClientTransportPlugin obfs3 exec /. . . /obfsproxy managed
UseBridges 1
Bridge obfs3 192.91.235.229:44444
Bridge:
BridgeRelay 1
ExtORPort auto
PublishServerDescriptor 0
ServerTransportPlugin obfs3 exec /.../obfsproxy managed
ServerTransportListenAddr obfs3 0.0.0.0:44444

the OP select our bridge as the entry node of circuits. Furthermore, we also modify the bridge's Tor code to delay and record cells, as presented in Section 4.

5.2 Data collection

Considering the popularity of websites, we select the 100 most popular websites from Alexa (http://www.alexa.com). However, some websites, such as Google, have multiple distinct country domains in different locations and occur many times on the list of top 100 sites. For these websites, we keep only the most popular one. To conduct experiments in the closed-world and open-world scenarios, we generate two datasets called *Data-closed* and *Data-open*. The *Data-closed* dataset consists of six cell-delay position schemes and every scheme contains the top 100 websites with 60 instances each. The *Data-open* dataset contains 1 instance each of 2000 non-monitored sites that are randomly selected from top 10 000 sites on Alexa. For the closed-world scenario, we select 40 instances each for training and 20 instances each for testing. To compare our method with previous passive attacks, we collect two additional datasets called *Data-passive* and *Data-passive-link* using the same monitor list. Cells are recorded in the *Data-passive* dataset at the bridge side while TCP traffic between OP and bridge is recorded in the *Data-passive-link* dataset. The information about four datasets is shown in Table 2.

5.3 Experimental results

We perform experiments in the closed-world scenario.

Table 2 Collected datasets.

Name	Content	Size	Training	Testing
Data-closed	Cell	100×60×6	100×40×6	100×20×6
Data-open	Cell	2000×1	0	2000×1
Data-passive	Cell	100×60	100×40	100×20
Data-passive-link	Packet	100×60	100×40	100×20

We believe that compromising the entry (bridge) node can bring more useful information to the attacker and increase the detection rate. To prove this assupmtion, we implement the passive website fingerprint attack on *Data-passive* and *Data-passive-link* datasets. We extract the same features mentioned in Section 4.6 and use the SVM classifier to predict the real destinations. For the *Data-passive-link* dataset, we convert packets into cells by rounding packet lengths to the closest multiple of 586 (512 bytes of a Tor cell and 74 bytes of a TLS header). The results are shown in Table 3. We can observe that the accuracy is increased by 0.1501 when the attacker compromises the bridge.

The previous work demonstrates that the more requests we delay, the higher accuracy we obtain[13]. However, multiple delays increase page loading time and seriously impact the user experience. According to our empirical experiments, the number of cell-delay positions is set to 4 or 5. Then, we execute two algorithms in Section 4 and obtain four delay schemes of different cell-delay positions, as shown in Table 4. The delay time is 500 ms except for the passive scheme, which is used as a baseline in this experiment.

Figure 9 illustrates the detection rates of four delay schemes. We use the SVM and k-NN classifiers on each scheme. For the k-NN algorithm, we vary the value of k from 1 to 10 on all schemes and the results are shown in Table 5. When k is 1, the classifier exhibits the best performance. Then, we set k to 1 in the following experiments. As we can observe in Fig. 9, by using the same classifier, actively delaying requests can improve the detection rate of the attack. The Active3 scheme achieves the highest accuracy regardless of what classifier is used. Therefore, in the following

Table 3 Accuracy of passive website fingerprinting attack on two datasets.

Scheme	Monitoring location	Dataset	Classifier	Accuracy
Passive1	Network layer	Data-passive-link	SVM	0.7360
Passive2	Bridge	Data-passive	SVM	0.8701

Table 4 Delay schemes of different delay positions.

Scheme	Delay positions	Delay time (ms)
Passive1	None	None
Active1	8, 9, 10, 15, 20	500
Active2	8, 9, 10, 20	500
Active3	8, 9, 10, 17, 27	500

Table 5 Detection rates of all schemes with different values of k.

Scheme	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$	$k=9$	$k=10$
Passive1	0.8595	0.7590	0.6865	0.6215	0.5635	0.5135	0.4690	0.4230	0.3820	0.3580
Active1	0.9626	0.9429	0.9237	0.9066	0.8869	0.8631	0.8439	0.8222	0.8015	0.7828
Active2	0.9705	0.9450	0.9297	0.9144	0.9016	0.8822	0.8629	0.8486	0.8328	0.8184
Active3	0.9745	0.9505	0.9295	0.9075	0.8940	0.8605	0.8445	0.8170	0.8010	0.7845
Active4	0.9602	0.9454	0.9255	0.9051	0.8903	0.8796	0.8612	0.8510	0.8362	0.8010
Active5	0.9864	0.9722	0.9626	0.9545	0.9419	0.9222	0.9010	0.8833	0.8742	0.8586
Active6	0.9737	0.9556	0.9394	0.9202	0.9015	0.8808	0.8621	0.8298	0.7793	0.7455

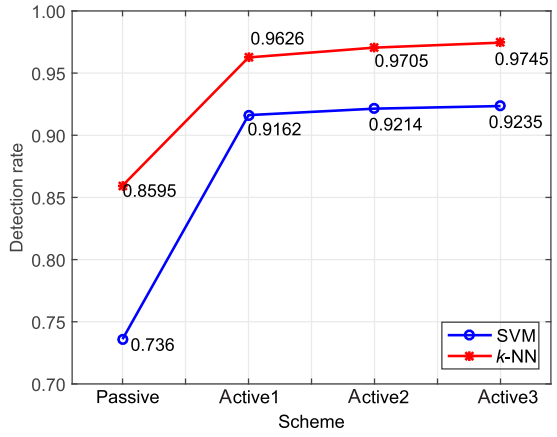

Fig. 9 Detection rates of four delay schemes.

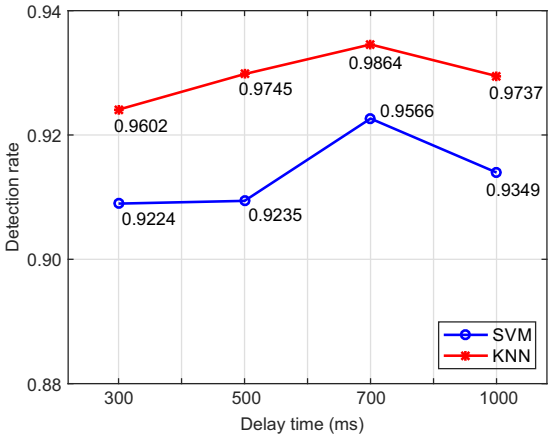

Fig. 10 Detection rates of different delay times.

experiments, we use cell-delay positions of the Active3 scheme.

Table 6 shows the different delay times of four schemes and the corresponding detection rates are shown in Fig. 10. The four schemes have the same delay positions, but the delay time is increased from 300 ms to 1000 ms. We can observe that the detection rates rise by increasing the delay time from 300 ms to 700 ms. However, when the delay time is increased to 1000 ms, the detection rates are slightly decreased. We deduce that excessive delay time may lead to close of the TCP streams and data retransmission, which can cause the accuracy to decrease. Therefore, the longer delay time is not positively related to the higher accuracy. In addition, when using the k-NN classifier on the Active5 scheme, we can obtain the highest accuracy of 0.9864. Thus, we select the k-NN algorithm as the classifier in our active attack.

To evaluate the practical feasibility of our active attack, we also perform experiments in the open-world scenario. Figure 11 illustrates the true and false positive rates in terms of delay time by using the k-NN classifier. Although the Active5 is still the optimal scheme, the False Positive Rate (FPR) remains high. Therefore, we change the value of k to decrease the false positive rate.

Figure 12 illustrates the results of FPR vs. True Positive Rate (TPR) while varying the number of k. When k is set to 5, we can obtain a 0.9096 true positive rate with a false positive rate of 0.039. Although varying k decreases the true positive rate, the false positive rate now is in an acceptable range in practical

Table 6 Delay schemes of different delay times.

Scheme	Delay position	Delay time (ms)
Active3	8, 9, 10, 17, 27	500
Active4	8, 9, 10, 17, 27	300
Active5	8, 9, 10, 17, 27	700
Active6	8, 9, 10, 17, 27	1000

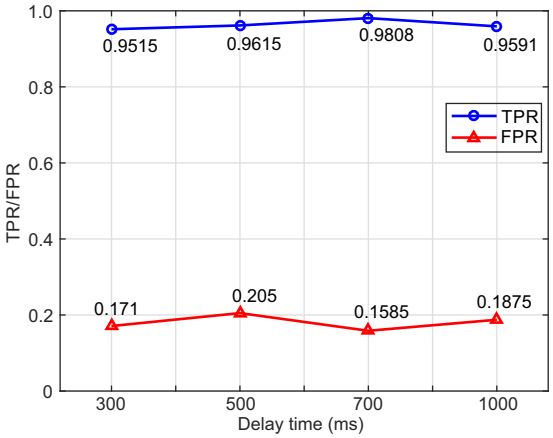

Fig. 11 True and false positive rates in terms of delay time.

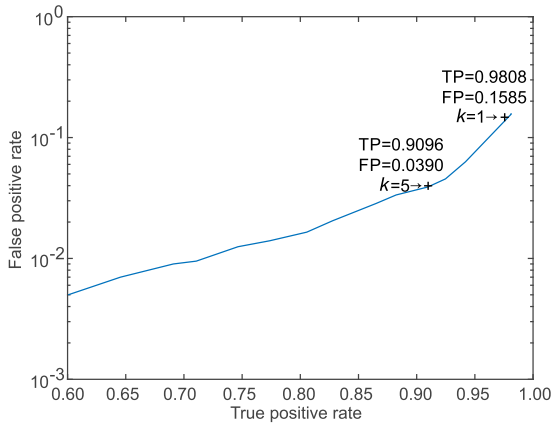

Fig. 12 True positive rates and false positive rates with different values of k.

scenarios. In addition, the corresponding accuracy in the closed-world scenario decreases to 0.9419.

6 Conclusion

Numberous of experiments demonstrate the effectiveness of the website fingerprinting attacks. However, in previous passive attacks, multiple HTTP web objects mixed in the encrypted traffic make extracted features obscure and unstable. In this paper, we proposed a novel active website fingerprinting attack performed at a controlled entry node. By actively delaying HTTP requests, we further improved the accuracy. To obtain a general scheme to select several appropriate cell-delay positions, we designed two algorithms based on statistical analysis and objective function optimization. Then, we compared the prediction ability of two classifiers in the closed-world scenario. By using k-NN, we could obtain the highest accuracy of 0.9864. We also performed experiments in the open-world scenario. When k is 5, we achieved a true positive rate of 0.9096 and a false positive rate of 0.039. In our future work, we will investigate the practical feasibility of our attack in the wild, such as in the task of filtering background traffic.

Acknowledgment

This work was partially supported by the National Key R&D Program of China (No. 2017YFB1003000), the National Natural Science Foundation of China (Nos. 61572130, 61320106007, 61632008, 61502100, 61532013, and 61402104); the Jiangsu Provincial Natural Science Foundation (No. BK20150637); the Jiangsu Provincial Key Technology R&D Program (No. BE2014603); the Qing Lan Project of Jiangsu Province, Jiangsu Provincial Key Laboratory of Network and Information Security (No. BM2003201); and the Key Laboratory of Computer Network and Information Integration of the Ministry of Education of China (No. 93K-9).

References

[1] Q. H. Liu, H. Shen, and Y. P. Sang, Privacy-preserving data publishing for multiple numerical sensitive attributes, *Tsinghua Sci. Technol.*, vol. 20, no. 3, pp. 246–254, 2015.

[2] Y. Wang, D. B. Xu, and F. Li, Providing location-aware location privacy protection for mobile location-based services, *Tsinghua Sci. Technol.*, vol. 21, no. 3, pp. 243–259, 2016.

[3] Thalmic labs, https://metrics.torproject.org/index.html, 2017.

[4] D. Herrmann, R. Wendolsky, and H. Federrath, Website fingerprinting: Attacking popular privacy enhancing technologies with the multinomial naïve-Bayes classifier, in *Proc. 2009 ACM Workshop on Cloud Computing Security*, Chicago, IL, USA, 2009, pp. 31–42.

[5] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, Website fingerprinting in onion routing based anonymization networks, in *Proc. 10^{th} Annual ACM Workshop on Privacy in the Electronic Society*, Chicago, IL, USA, 2011, pp. 103–114.

[6] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson, Touching from a distance: Website fingerprinting attacks and defenses, in *Proc. 2012 ACM Conf. Computer and Communications Security*, Raleigh, NC, USA, 2012, pp. 605–616.

[7] W. Yu, X. W. Fu, S. Graham, D. Xuan, and W. Zhao, DSSS-based flow marking technique for invisible traceback, in *IEEE Symp. Security and Privacy*, Berkeley, CA, USA, 2007, pp. 18–32.

[8] Z. Ling, J. Z. Luo, W. Yu, X. W. Fu, D. Xuan, and W. J. Jia, A new cell-counting-based attack against Tor, *IEEE/ACM Trans. Netw.*, vol. 20, no. 4, pp. 1245–1261, 2012.

[9] M. Yang, J. Z. Luo, Z. Ling, X. W. Fu, and W. Yu, De-anonymizing and countermeasures in anonymous communication networks, *IEEE Commun. Mag.*, vol. 53, no. 4, pp. 60–66, 2015.

[10] T. Wang and I. Goldberg, Improved website fingerprinting on Tor, in *Proc.12^{th} ACM Workshop on Workshop on Privacy in the Electronic Society*, Berlin, Germany, 2013, pp. 201–212.

[11] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, Effective attacks and provable defenses for website fingerprinting, in *Proc. 23^{rd} USENIX Conf. Security Symposium*, San Diego, CA, USA, 2014, pp. 143–157.

[12] A. Panchenko, F. Lanze, A. Zinnen, M. Henze, J. Pennekamp, K. Wehrle, and T. Engel, Website fingerprinting at internet scale, in *Proc. 23^{rd} Internet Society (ISOC) Network and Distributed System Security Symp.*, San Diego, CA, USA, 2016.

[13] G. F. He, M. Yang, X. D. Gu, J. Z. Luo, and Y. Y. Ma, A novel active website fingerprinting attack against tor anonymous system, in *Proc. 2014 IEEE 18^{th} Int. Conf.*

Computer Supported Cooperative Work in Design CSCWD), Hsinchu, China, 2014, pp. 112–117.
[14] J. Hayes and G. Danezis, Better open-world website fingerprinting, arXiv preprint arXiv: 1509.00789, 2015.
[15] M. Juarez, S. Afroz, G. Acar, C. Diaz, and R. Greenstadt, A critical evaluation of website fingerprinting attacks, in *Proc. 2014 ACM SIGSAC Conf. Computer and Communications Security*, Scottsdale, AR, USA, 2014, pp. 263–274.
[16] T. Wang and I. Goldberg, On realistically attacking tor with website fingerprinting, *Proc. Priv. Enhanc. Technol.*, vol. 2016, no. 4, pp. 21–36, 2016.
[17] Libevent–an event notification library, http://libevent.org/, 2017.
[18] V. Pappas, E. Athanasopoulos, S. Ioannidis, and E. P. Markatos, Compromising anonymity using packet spinning, in *Information Security*, T. C. Wu, C. L. Lei, V. Rijmen, and D. T. Lee, eds. Berlin, Heidelberg: Springer, 2008, pp. 161–174.
[19] Tor protocol specification, https://svn.torproject.org/svn/tor/branches/hidserv-perf/doc/spec/tor-spec.txt, 2017.
[20] Tor directory protocol, http://tor.eff.org/svn/trunk/doc/spec/dir-spec.txt, 2015.
[21] K. Bauer, D. McCoy, D. C. Grunwald, T. Kohno, and D. Sicker, Low-resource routing attacks against anonymous systems, Tech. Rep. CU-CS-1025-07, University of Colorado Boulder, CO, USA, 2007.
[22] Z. Ling, J. Z. Luo, W. Yu, M. Yang, and X. W. Fu, Tor bridge discovery: Extensive analysis and large-scale empirical evaluation, *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 7, pp. 1887–1899, 2015.
[23] Z. Ling, J. Z. Luo, W. Yu, M. Yang, and X. W. Fu, Extensive analysis and large-scale empirical evaluation of tor bridge discovery, in *Proc. IEEE INFOCOM*, Orlando, FL, USA, 2012, pp. 2381–2389.
[24] C. C. Chang and C. J. Lin, LIBSVM: A library for support vector machines, *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, p. 27, 2011.
[25] PLANETLAB, https://www.planet-lab.org/, 2017.
[26] Tor project obfsproxy, https://www.torproject.org/projects/obfsproxy.html.en, 2015.

Ming Yang received the PhD degree in computer science from Southeast University, China, in 2007. Currently, he is an associate professor at the School of Computer Science and Engineering in Southeast University, Nanjing, China. His research interests include network security and privacy. Dr. Yang is a member of CCF and ACM, as well as Deputy Director of Key Laboratory of Computer Network and Information Integration, Ministry of Education.

Xiaodan Gu received the BS degree from Beijing Institute of Technology in 2009. She is currently working toward the PhD degree in computer science at Southeast University, China. Her research interests include network security and privacy.

Changxin Yin is currently working toward the MS degree at Southeast University, China. She received the BS degree from Zhengzhou University in 2014. Her research interests include network security and privacy.

Zhen Ling received the PhD degree in computer science from Southeast University, China in 2014. Currently, he is an associate professor at the School of Computer Science and Engineering in Southeast University, Nanjing, China. His research interests include smartphone security, network security, privacy, and forensics. He won ACM China Doctoral Dissertation Award and China Computer Federation (CCF) Doctoral Dissertation Award, in 2014 and 2015, respectively.

Junzhou Luo received the BS degree in applied mathematics and the MS and PhD degrees in computer network from Southeast University, China, in 1982, 1992, and 2000, respectively. He is a full professor in the School of Computer Science and Engineering, Southeast University, Nanjing, China. His research interests are next generation network architecture, network security, cloud computing, and wireless LAN. He is a member of the IEEE Computer Society and co-chair of IEEE SMC Technical Committee on Computer Supported Cooperative Work in Design, and a member of the ACM and chair of ACM Sigcomm China.