IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 28, 2020, accepted July 8, 2020, date of publication July 21, 2020, date of current version July 30, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3010756

Tracing Website Attackers by Analyzing Onion Routers' Log Files

YINAN PEI AND KAZUMASA OIDA, (Member, IEEE)
Graduate School of Engineering, Fukuoka Institute of Technology, Fukuoka 811-0295, Japan
Corresponding author: Kazumasa Oida (oida@fit.ac.jp)

ABSTRACT The Onion Router (Tor) is one of the major network systems that provide anonymous communication and censorship circumvention. Tor enables its users to surf the Internet, chat, and send messages anonymously; however, cyber attackers also exploit the system for circumventing criminal activity detection. Recently, various approaches that prevent or mitigate abuse of Tor have been proposed in the literature. This paper, which presents one of the approaches, addresses an IP traceback problem. In our model, onion routers that voluntarily participate in attacker tracing detect attack packets (packets carrying an attacker's code or data) recorded in the log files by sharing necessary information with an attacked server over an Ethereum blockchain network. The detection algorithm in this paper uses the statistics of packet travel and relay times and outputs attack-packet candidates. The proposed method attaches a reliability degree to each candidate, which is based on the upper bounds of its Type I and II error rates. A smart contract running on the blockchain network ranks the detection results from onion routers according to the reliability degrees.

INDEX TERMS Ethereum, IP traceback, log analysis, network security, Tor.

I. INTRODUCTION

The Onion Router (Tor) [1] is a widely used overlay network that provides low-latency anonymous communication for transmission control protocol (TCP) applications and helps circumvent various censorship measures. According to Tor Metrics [2], the Tor network currently consists of more than 6,000 onion routers, has millions of directly connecting users, and carries hundreds of Gbit/s. Tor, however, has been abused by illegal services [3], [4], such as the infamous Silk Road [5] and the CryptoLocker ransomware command and control (C&C) servers [6]. It was reported in [7] that some onion routers are malicious and perform man-in-the-middle (MITM), structured query language (SQL) injection, and cross-site scripting (XSS) attacks.

Recently, various approaches to maintain the health of the Tor network have been discussed in the literature [7]–[11]. Previous studies were focused on blocking traffic, routers, or hidden servers that are considered to be malicious. This paper, in contrast, considers an Internet protocol (IP) traceback problem over the Tor network to search for the criminals who triggered an attack. In our model, volunteer onion routers investigate the IP address of the attacker's machine in conjunction with the attacked servers. We expect that successful investigations enabled by our scheme will be a strong deterrent to attacks even if it is only partially deployed in the Tor network. IP traceback problems have been intensively studied, in particular for countering denial-of-service (DoS) and distributed denial-of-service (DDoS) attacks [12]. However, to the best of our knowledge, in no papers, has a means of detecting the actual attack source in the Tor network been discussed.

According to [13], the majority of Tor research has been devoted to *deanonymization*, the design of a breaching strategy. Deanonymization based on traffic analysis is somewhat similar to our approach. From the perspective of attackers, the objective of deanonymization is to maximize the success rate of linking a source and a destination of any communication. Based on the perspective of a criminal investigation, our approach logically narrows down the candidates for *attack packets*, packets carrying the attacker's code or data, based on the evidence remaining on the victim server. Thus, our approach may not detect a single candidate but it does significantly decrease the detection error rates. The premise of our approach is different from that of traffic analysis-based deanonymization in that:

1) The approach is focused on a specified communication, where a partner of the communication is a victim

The associate editor coordinating the review of this manuscript and approving it for publication was Hualong Yu.

server, whereas adversaries in general attack communications indiscriminately.
2) To minimize the effect on the Tor system and users, the approach modifies neither the Tor software nor the Tor protocol, whereas adversaries may do so.
3) The approach can receive support from the victim server, whereas adversaries would never obtain this support.
4) The approach considers the privacy of Tor users, whereas adversaries do not.

In our model, attacked servers and routers form an Ethereum blockchain network [14], in which a smart contract, called attacker tracing (AT), acts as an attacker tracing manager that receives incident reports from attacked servers and detection reports from routers. Each detection result includes a reliability degree, which is based on the upper bounds of Type I and II error rates. We employ blockchain technologies, which check for log data forgery and maintain the tracing processes in their entirety to ensure no leakage of Tor users' private information occurs. The process of tracing attackers consists of two phases: learning and detection. Routers collect samples of packet travel and relay times in the learning phase. The samples are then employed in the detection phase to identify attack packets in the router's log file.

The paper is organized as follows. Section II provides the background of the study and Section III presents related work. Section IV describes the AT model, and Section V details the approach for detecting the attacker's IP address. Section VI shows the experimental environment and results. Section VII analyzes the experimental data to derive the reliability of the detection results. Section VIII discusses the implications of the findings, and Section IX concludes the paper.

II. BACKGROUND

A. TOR

Tor is a low-latency anonymity network based on a concept called *onion routing*, which operates as follows. A client who installs an onion proxy, an interface between a client and the Tor network, downloads onion router information from a directory server and chooses three routers to establish a circuit. The first, second, and third routers are known as the entry, middle, and exit routers, respectively. Packets from the client to a server pass through the circuit.

The client sends packets that are encrypted with multiple layers of encryption using keys negotiated between the client and each router. As packets travel along the circuit, each router strips off one layer of encryption. This layered encryption ensures that each router knows the identities only of routers that are directly connected in the circuit. According to [15], to counter traffic correlation attacks, no two routers are chosen from the same family group to create a circuit. Sometimes, a bridge is introduced as a hidden entry router to resist censorship further.

B. ETHEREUM

Ethereum is the second-largest cryptocurrency platform on which users broadcast transactions (data packets signed with their private keys). Ethereum and Bitcoin [16] are similar in that peer-to-peer (P2P) technology maintains a blockchain, a growing list of transaction records that are linked using cryptography, through the competition of solving computationally intensive problems. While Bitcoin blockchains are concerned only with transactions between user wallets, Ethereum blockchains present decentralized computing environments called Ethereum Virtual Machines (EVMs), on which smart contracts (stateful decentralized applications) can run. Two types of accounts exist in Ethereum: user accounts controlled by users and contract accounts controlled by smart contracts. Smart contracts behave in the same manner as autonomous agents.

III. RELATED WORK

Since the inception of the Tor system, many deanonymization approaches based on traffic analysis have been developed. Traffic-analysis attack methods can be classified into two groups: passive and active. In the active (passive) method, attackers (do not) alter traffic patterns. There exist two well-known passive traffic-analysis attacks. First, the Website fingerprinting attacks [17], [18] infer which Webpage a client is visiting by identifying traffic patterns that are unique to the Webpage. This method is considered effective when `.onion` sites are distinguished from regular sites [17]. In this paper, it was not assumed that the attacked servers generate characteristic traffic. Second, the end-to-end confirmation (or traffic correlation) attacks [19]–[22] correlate the traffic of a flow over an input link with that over an output link. The approach presented in this paper detects two packets, a request and a response, that are specific (i.e., they are used for an attack), not a flow of unspecified many packets. An additional attack exists called deep packet inspection [23], which inspects in detail the data being sent to characterize the traffic, protocol, and application. Our approach also inspects packet fields, such as the TCP push flag and time-stamps (TS) option [24], to narrow down the attack-packet candidates.

Meanwhile, approaches exist that prevent or mitigate abuse of Tor. In [8], the author discussed the countermeasures that prevent attempts to reach the `.onion` addresses of C&C servers, such as that of *mevade* running as a Tor onion service. The authors of [9] proposed a system called TorWard, which was designed to discover and classify malicious traffic over Tor using an intrusion detection system (IDS). In [7], the effectiveness of honey onions (HOnions) servers, which detect and identify misbehaving and snooping hidden service directories, was reported. Torpolice, a privacy-preserving access control framework, enables service providers to define access policies against requests coming from Tor [10]. Meek is a pluggable transport that allows Tor traffic to be hidden in the background traffic. To enhance the possibility of

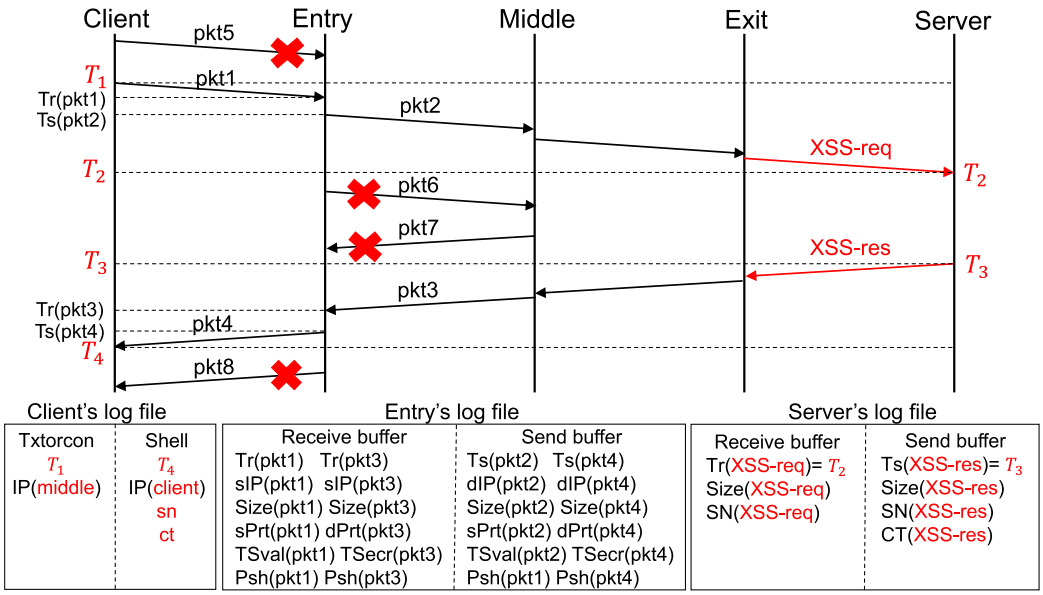

FIGURE 1. The client makes an XSS attack against the server through the Tor network. The aim of the study was to detect packets pkt1, pkt2, pkt3, and pkt4 that carry the XSS-req or XSS-res data by removing unrelated packets pkt5, pkt6, pkt7, and pkt8 from the candidates.

monitoring illegal activities over Tor, a meek-based traffic identification method was proposed in [11]. These prevention or mitigation approaches are focused on detecting malicious traffic or Tor components, whereas this paper considers the problem of tracing back attack packets to their origins.

The IP traceback problem is defined as identifying the actual source of any packet transmitted across the Internet [25]. Various approaches have been proposed [26], [27], and recent studies were focused especially on DDoS attacks using Internet of Things (IoT) devices [12]. Currently, there are six main categories of IP traceback methods [27]: link testing [28], messaging [29], marking [30], logging [31], overlay [32], and pattern analysis [33]. None of these can be applied to the model in this paper for three reasons. First, because messages in the Tor network are encrypted with multiple layers of encryption, it is impossible to associate packet contents observed at onion routers with those observed at the destination host. Accordingly, marking, messaging, and logging approaches that use packet contents for identification are useless. Second, Tor is not a network managed by one organization, and thus approaches that assume cooperation between many routers, such as link testing and overlay, cannot be employed. Third, approaches requiring many attack packets, such as link testing and pattern analysis, are not available. Our model assumes attackers send one message and receive one response.

IV. ATTACKER TRACING MODEL

A. PROPOSED MODEL

Fig. 1 illustrates the proposed model. The client makes an XSS attack against the server through the Tor network. The server receives an XSS-req and sends an XSS-res, where an XSS-req (XSS-res) means an HTTP request (response) message used by the XSS attack. Note that, except for XSS-req and XSS-res, the TCP payloads of all the packets in Fig. 1 are encrypted. Packets pkt1 and pkt2 (pkt3 and pkt4) carry information of the XSS-req (XSS-res), and therefore, pkt1, pkt2, pkt3, and pkt4 are attack packets. This section describes a model for the case of an XSS attack, but the descriptions of other attack cases, such as SQL injection and cross-site request forgeries (CSRF), are very similar.

Hereinafter, we use the term cooperators to indicate routers that participate in detecting the source of the XSS-req. The detection is easy if all three routers (entry, middle, and exit) in Fig. 1 are cooperators. The detection of the attacker's address requires that at least the entry router is a cooperator (the attacker's address is the source IP address of pkt1). This paper considers the case where only the entry router is a cooperator. Thus, the objective of this study was to find attack packets (pkt1, pkt2, pkt3, pkt4) in the entry's log file in cooperation with the attacked server, and we call this cooperative action *attacker tracing*.

Fig. 1 shows all the log data required in our approach. Throughout this paper, the following notations are used.

- $T_s(p)$ ($T_r(p)$): time when packet p is sent (received).
- $dIP(p)$ ($sIP(p)$): destination (source) IP address of packet p.
- $dPrt(p)$ ($sPrt(p)$): destination (source) TCP port number of packet p.
- $TSval(p)$ ($TSecr(p)$): time-stamp value (echo reply) of packet p.
- $Psh(p)$: TCP push flag of packet p.
- $Size(p)$: IP packet length of packet p.
- $IP(m)$: IP address of machine m.

- $SN(p)$ ($CT(p)$): sequence number (current time) of packet p.
- sn (ct): sequence number (current time) inserted in an HTTP message by the client (server).

Four times T_1, T_2, T_3, and T_4 in Fig. 1 are defined as

$$T_1 := T_s(pkt1), \tag{1}$$
$$T_2 := T_r(XSS\text{-}req), \tag{2}$$
$$T_3 := T_s(XSS\text{-}res), \tag{3}$$
$$T_4 := T_r(pkt4). \tag{4}$$

The entry router and server store the timestamp and a part of the content of every incoming and outgoing packet. As shown in Fig. 1, the amount of data in the entry is not large. The log data of the client are outputs of the shell script (that executes XSS attacks) and Python code `stream_circuit_logger.py` in the txtorcon library [34], which provides live state information about Tor circuits and routers. The client's log data are used to obtain correct attack packets (pkt1,pkt2,pkt3,pkt4).

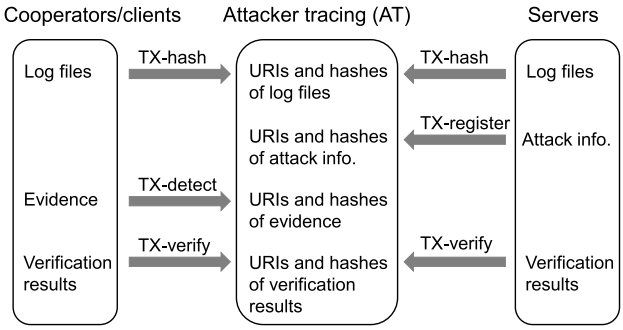

FIGURE 2. Cooperators, clients, and servers periodically send hashes of log files to attacker tracing (AT). AT receives `TX-register` reporting attack information, `TX-detect` reporting the evidence of detection, and `TX-verify` reporting the verification result of the evidence.

B. SMART CONTRACT

Assume that all cooperators and attacked servers belong to an Ethereum network, in which a smart contract called attacker tracing (AT) is working. Fig. 2 shows the interactions among Ethereum nodes. Every certain period of time, every Ethereum node makes a log file shown in Fig. 1 and issues a `TX-hash` transaction that contains the uniform resource identifier (URI) and the hash value of the log file. To avoid privacy violations, the IP addresses in the log file must be replaced with dummy addresses (e.g., the hashed values). After an attack has occurred, as shown in Fig. 2, the victim server publishes a `TX-register` that carries the URI and the hash value of the attack information file, which includes T_2, T_3, and a TCP port number of the server for sampling. Each cooperator then selects a client that generates HTTP packets to the TCP port to sample travel times etc. To prevent log file forgery, the client must send `TX-hash` transactions during the sampling period.

After calculating attack-packet candidates, each cooperator publishes a `TX-detect` that includes the URI and the hash value of the evidence, which consists of four timestamps $T_r(pkt1)$, $T_s(pkt2)$, $T_r(pkt3)$, and $T_s(pkt4)$ (which represent an attack-packet candidate) and reliability degrees of all candidates. The attacked server and cooperators can verify the evidence and send `TX-verify` transactions, including the URIs and the hash values of their verification results. The most reliable candidates should be verified first. In the verification process, using log data that each cooperator or the server locally possesses, it is checked that the evidence is not logically inconsistent. To avoid false positive and false negative errors, the evidence must be carefully verified from many different viewpoints.

The following explicates why the Ethereum technologies are used.

1) They automatically verify and maintain the integrity of share data (the log files, attack information, evidence, and verification results). When AT receives a transaction, it always verifies the hash values in the transaction.
2) They help prevent a server from forging a cyberattack incident. Contract AT checks two signatures in a `TX-register`: the digital signature of the server and the signature of a third party who guarantees the correctness of the attack information.
3) They allow all cooperators to reexamine at any time the tracing processes of all past incidents in their entirety. Cooperators can always check whether privacy violations occurred.

V. DETECTION PROCESS

A. LEARNING PHASE

The detection process starts after a `TX-register` enters an Ethereum block. It consists of learning and detection phases. In the learning phase, statistical features related to attack packets are estimated. To this end, a client selected by a cooperator performs simulated attacks such that the client periodically transmits XSS-req (to the TCP port number in the `TX-register`) using the cooperator as his/her entry router. The XSS-req and XSS-res messages should be identical to those used by the attacker. After the simulated attacks, the entry router (i.e., the cooperator) gathers log data of the server and client, all of which are depicted in Fig. 1, using URIs in `TX-hash` transactions.

Next, the entry router solves the following problem for each attack, i.e., for each 4-tuple (T_1, T_2, T_3, T_4).

Problem 1: Find all 4-tuples (pkt1,pkt2,pkt3,pkt4) in the entry's log file that satisfy the following conditions:

1. $T_1 < T_r(pkt1) < T_s(pkt2) < T_2 < T_3 < T_r(pkt3) < T_s(pkt4) < T_4$.
2. $Size(XSS\text{-}req) \leq Size(pkt1)$, $Size(XSS\text{-}req) \leq Size(pkt2)$, $Size(XSS\text{-}res) \leq Size(pkt3)$, $Size(XSS\text{-}res) \leq Size(pkt4)$.

3. $sIP(pkt1) = dIP(pkt4) = IP(client)$,
   $dIP(pkt2) = sIP(pkt3) = IP(middle)$.
4. $sPrt(pkt1) = dPrt(pkt4)$,
   $sPrt(pkt2) = dPrt(pkt3)$.
5. $TSval(pkt1) = TSecr(pkt4)$,
   $TSval(pkt2) = TSecr(pkt3)$.
6. $Psh(pkt1) = Psh(pkt2) = Psh(pkt3) = Psh(pkt4) = 1$.

The following explains the conditions in Problem 1. All of them are necessary conditions that attack packets must satisfy. It is clear in Fig. 1 that attack packets pkt1-pkt4 satisfy Condition 1 and packets pkt5-pkt8 marked with red crosses are excluded by Condition 1. Condition 2 holds according to the Tor protocol specification [35]. Condition 3 implies that attack packets pass through a circuit to reach the server. Condition 4 compares the TCP port numbers. We do not check $dPrt(pkt1) = sPrt(pkt4)$ and $dPrt(pkt2) = sPrt(pkt3)$, because the port numbers of the entry and middle routers are typically fixed at 443. Condition 5 checks the TCP TS option to connect pkt1 with pkt4 and pkt2 with pkt3. Fig. 10 in Appendix A exemplifies the values of the option field. The TS option was originally used for round-trip time measurement [24]. Condition 6 requires that the TCP push flag be set in "`HTTP GET`" and "`HTTP/1.1 200 OK`" packets [36].

B. PROGRAM $\mathcal{P}1$

Program $\mathcal{P}1$, which solves Problem 1, first removes abnormal communication from the log file, which corresponds to the case where the client (server) received (sent) an HTTP status code that is not 200. The program then determines T_1, T_2, T_3, T_4, and $IP(middle)$ for each attack. Address $IP(middle)$ associated with T_1 is entered in the client's log file (see Fig. 1). A pair of (T_1, T_4) in the client's log file is linked with a pair of (T_2, T_3) in the server's log file by examining whether sequence number sn uniquely assigned to each attack satisfies

$$SN(XSS\text{-}req) = SN(XSS\text{-}res) = sn. \tag{5}$$

The program also checks whether current time ct inserted by the server satisfies

$$CT(XSS\text{-}res) = ct, \tag{6}$$

which connects T_3 and T_4. Appendix A exemplifies (5) and (6) by showing samples of $SN(XSS\text{-}req)$, $SN(XSS\text{-}res)$, $CT(XSS\text{-}res)$, sn, and ct obtained in the experiment.

Cases occur where more than one XSS-req or XSS-res packets satisfy (5) or (6) because of TCP retransmission. In these cases, the program checks whether the relation

$$T_1 < T_2 < T_3 < T_4 \tag{7}$$

holds. For each attack, there may exist multiple 4-tuples (T_1, T_2, T_3, T_4) that satisfy (5)–(7). In this case, $\mathcal{P}1$ calculates solutions for all 4-tuples (T_1, T_2, T_3, T_4). Let u^1 be the number of solutions for an attack. "$u^1 = 1$" indicates that the attack packets are successfully detected. Only unique solutions ($u^1 = 1$) are used to calculate statistics.

C. STATISTICS

Unique solutions (pkt1,pkt2,pkt3,pkt4) in the learning phase are used to obtain travel times (T_{req} and T_{res}) and relay times (T_{12} and T_{34}) defined by

$$T_{req} := T_2 - T_s(pkt2), \tag{8}$$
$$T_{res} := T_r(pkt3) - T_3, \tag{9}$$
$$T_{12} := T_s(pkt2) - T_r(pkt1), \tag{10}$$
$$T_{34} := T_s(pkt4) - T_r(pkt3). \tag{11}$$

Note in Fig. 1 that T_{req} is the packet travel time from the entry router to the server. The entry router then calculates the means (standard deviations) of T_{req}, T_{res}, T_{12}, and T_{34}, which are denoted by m_{req}, m_{res}, m_{12}, and m_{34} (s_{req}, s_{res}, s_{12}, and s_{34}), respectively.

The entry router also calculates the linear regression parameters for samples $\{(T_{req}, T_{res})\}$. If T_{req} and T_{res} are considered x and y coordinate values, we have two least-squares regression lines given by

$$y = a_{res}x + b_{res}, \tag{12}$$
$$x = a_{req}y + b_{req}, \tag{13}$$

where a_{res} and b_{res} (a_{req} and b_{req}) are regression coefficients for predicting T_{res} (T_{req}). If line (12) (line (13)) provides a good prediction, then e_{res} (e_{req}), the square root of the mean square errors for predicting T_{res} (T_{req}), is small. In other words, T_{req} (T_{res}) is close to $\tilde{T}_{req}$ ($\tilde{T}_{res}$), where

$$\tilde{T}_{res} := a_{res}(T_2 - T_s(pkt2)) + b_{res}, \tag{14}$$
$$\tilde{T}_{req} := a_{req}(T_r(pkt3) - T_3) + b_{req}. \tag{15}$$

D. DETECTION PHASE

This phase does not allow cooperators to access the information of the client (the adversary's machine) but does allow them to utilize the statistics described in Section V-C. A cooperator detects attack packets that correspond to (T_2, T_3) in `TX-register` by solving the following problem.

Problem 2: Given α_1, α_2, and α_3, find all 4-tuples (pkt1,pkt2,pkt3,pkt4) in the cooperator's log file that satisfy the following conditions:

1. $T_1' < T_r(pkt1) < T_s(pkt2) < T_2$
   $< T_3 < T_r(pkt3) < T_s(pkt4) < T_4'$, where
   $$T_1' = T_2 - (m_{req} + m_{12}) - \alpha_1(s_{req} + s_{12}), \tag{16}$$
   $$T_4' = T_3 + (m_{res} + m_{34}) + \alpha_1(s_{res} + s_{34}). \tag{17}$$
2. $Size(XSS\text{-}req) \le Size(pkt1)$,
   $Size(XSS\text{-}req) \le Size(pkt2)$,
   $Size(XSS\text{-}res) \le Size(pkt3)$,
   $Size(XSS\text{-}res) \le Size(pkt4)$.
3. $sIP(pkt1) = dIP(pkt4)$,
   $dIP(pkt2) = sIP(pkt3)$.
4. $sPrt(pkt1) = dPrt(pkt4)$,
   $sPrt(pkt2) = dPrt(pkt3)$.
5. $TSval(pkt1) = TSecr(pkt4)$,
   $TSval(pkt2) = TSecr(pkt3)$.

6. $Psh(pkt1) = Psh(pkt2) = Psh(pkt3) = Psh(pkt4) = 1$.
7. $-\alpha_2\ e_{req} < (T_2 - T_s(pkt2)) - \tilde{T}_{req} < \alpha_2\ e_{req}$.
8. $-\alpha_2\ e_{res} < (T_r(pkt3) - T_3) - \tilde{T}_{res} < \alpha_2\ e_{res}$.
9. $-\alpha_3\ s_{12} < (T_s(pkt2) - T_r(pkt1)) - m_{12} < \alpha_3\ s_{12}$.
10. $-\alpha_3\ s_{34} < (T_s(pkt4) - T_r(pkt3)) - m_{34} < \alpha_3\ s_{34}$.

Note that $\tilde{T}_{req}$ and $\tilde{T}_{res}$ in Conditions 7 and 8 are defined in (14) and (15). Conditions 1–6 are the same as those in Problem 1, except that T_1 and T_2 are replaced with T_1' and T_2', respectively, and $IP(client)$ and $IP(middle)$ do not appear in Condition 3 (because they are in the client's log file). The calculated statistics are used in Conditions 1 and 7–10.

In Condition 1, (16) and (17) calculate T_1' and T_2' using the means and standard deviations of travel and relay times. Positive real number α_1 in (16) and (17) controls the range in which the correct solution (the attack packets) should exist. A too small (large) α_1 results in $u^2 = 0$ ($u^2 \geq 2$), where u^2 is the number of solutions to Problem 2. Similarly, positive real number α_2 (α_3) in Conditions 7 and 8 (9 and 10) controls the distance from the least-squares lines (the mean relay time) that the correct solution should satisfy. Conditions 7–10 are optional.

E. PROGRAM $\mathcal{P}2$

Program $\mathcal{P}2$, which solves Problem 2, operates according to Algorithm 1. In the algorithm, $R(T_1', T_4')$ ($S(T_1', T_4')$) is a set of all packets p that are received (sent) by the cooperator between the times T_1' and T_4' and satisfy $Psh(p) = 1$, where T_1' and T_4' are given in (16) and (17). Let B_1, B_2, B_3, and B_4 be the candidate sets of attack packets pkt1, pkt2, pkt3, and pkt4, respectively. They are created in lines 2–19 by verifying Condition 2 and consulting cache $A_{onionoo}$ that contains onion routers' IP addresses. $A_{onionoo}$ is maintained by sending a request to the Onionoo server [37] and is used to determine whether a packet came from an onion router. Note that program $\mathcal{P}1$ does not require the cache, because $\mathcal{P}1$ needs to consider only packets from/to the client and the middle router, and their addresses $IP(client)$ and $IP(middle)$ are given.

Lines 20–39 create B_{14} and B_{23}, which are the candidate sets of attack packet pairs (pkt1,pkt4) and (pkt2,pkt3), respectively, by checking Conditions 3–5, 7, and 8. $B_1 \times B_4$ in line 20 denotes Cartesian product $\{(p, q) | p \in B_1, q \in B_4\}$. Finally, lines 40–48 create B_{1234}, which is the set of solutions to Problem 2, by finding two pairs $(p, q) \in B_{14}$ and $(u, v) \in B_{23}$ that satisfy Conditions 1, 9, and 10.

Algorithm 1 Solving Problem 2

Input: $R(T_1', T_4')$, $S(T_1', T_4')$

1: $B_1 = B_2 = B_3 = B_4 = B_{14} = B_{23} = B_{1234} = \emptyset$
2: **for all** $p \in R(T_1', T_4')$ **do**
3: **if** $sIP(p) \in A_{onionoo}$ **then**
4: **if** $Size(XSS\text{-}res) \leq Size(p)$ **then** $B_3 \leftarrow B_3 \cup \{p\}$
5: **end if**
6: **else**
7: **if** $Size(XSS\text{-}req) \leq Size(p)$ **then** $B_1 \leftarrow B_1 \cup \{p\}$
8: **end if**
9: **end if**
10: **end for**
11: **for all** $q \in S(T_1', T_4')$ **do**
12: **if** $dIP(q) \in A_{onionoo}$ **then**
13: **if** $Size(XSS\text{-}req) \leq Size(q)$ **then** $B_2 \leftarrow B_2 \cup \{q\}$
14: **end if**
15: **else**
16: **if** $Size(XSS\text{-}res) \leq Size(q)$ **then** $B_4 \leftarrow B_4 \cup \{q\}$
17: **end if**
18: **end if**
19: **end for**
20: **for all** $(p, q) \in B_1 \times B_4$ **do**
21: **if** $TSval(p) = TSecr(q)$ **then**
22: **if** $sIP(p) = dIP(q)$ and $sPrt(p) = dPrt(q)$ **then**
23: $B_{14} \leftarrow B_{14} \cup \{(p, q)\}$
24: **end if**
25: **end if**
26: **end for**
27: **for all** $(u, v) \in B_2 \times B_3$ **do**
28: **if** $TSval(u) = TSecr(v)$ **then**
29: **if** $dIP(u) = sIP(v)$ and $sPrt(u) = dPrt(v)$ **then**
30: $\tilde{x} \leftarrow a_{req}(T_r(v) - T_3) + b_{req}$
31: **if** $T_2 - T_s(u) - \tilde{x} \in (-\alpha_2\ e_{req}, \alpha_2\ e_{req})$ **then**
32: $\tilde{y} \leftarrow a_{res}(T_2 - T_s(u)) + b_{res}$
33: **if** $T_r(v) - T_3 - \tilde{y} \in (-\alpha_2\ e_{res}, \alpha_2\ e_{res})$ **then**
34: $B_{23} \leftarrow B_{23} \cup \{(u, v)\}$
35: **end if**
36: **end if**
37: **end if**
38: **end if**
39: **end for**
40: **for all** $((p, q), (u, v)) \in B_{14} \times B_{23}$ **do**
41: **if** $T_r(p) < T_s(u) < T_2 < T_3 < T_r(v) < T_s(q)$ **then**
42: **if** $T_s(u) - T_r(p) - m_{12} \in (-\alpha_3\ s_{12}, \alpha_3\ s_{12})$ **then**
43: **if** $T_s(q) - T_r(v) - m_{34} \in (-\alpha_3\ s_{34}, \alpha_3\ s_{34})$ **then**
44: $B_{1234} \leftarrow B_{1234} \cup \{(p, u, v, q)\}$
45: **end if**
46: **end if**
47: **end if**
48: **end for**

Output: B_{1234}

VI. EXPERIMENTS

A. EXPERIMENTAL ENVIRONMENT

Fig. 3 illustrates the experimental system, in which Client i, $i = 1, 2, \ldots, 5$, communicates with Server i *via* a bridge (a private entry router). A bridge is used for controlling the volume of traffic passing through an entry router. The bridge, Client 1, and Server 1 execute the network time protocol (NTP) [38] for clock synchronization and tcpdump [39] for recording the timestamps and contents of packets.

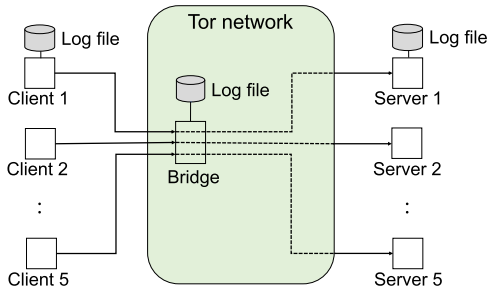

FIGURE 3. Client i, $i = 1, 2, \ldots, 5$, communicates with Server i *via* the bridge that detects packets exchanged between Client 1 and Server 1. Clients 2–5 generate traffic that introduces noise into the bridge's log file.

We conducted three experiments and for each experiment, we obtained two datasets: $\mathcal{L}$ and $\mathcal{E}$. $\mathcal{L}$ and $\mathcal{E}$ both contain log data of Client 1, Server 1, and the bridge for the sampling period of approximately five hours. Dataset $\mathcal{L}$ was used for calculating the statistics of travel and relay times, and $\mathcal{E}$ was used for evaluating our approach. To allow a fair evaluation, learning and evaluation datasets are different.

TABLE 1. Server i, $i = 1, \ldots, 5$, the TCP payload length (bytes) of the HTTP response from Server 1, and the client access intervals (in seconds). Symbol "-" denotes the server is not used.

Server	Exp. 1	Exp. 2	Exp. 3
1: getdate.php	265 B, 10 s	1,208 B, 10 s	265 B, 10 s
2: www.yahoo.co.jp	5 s	5 s	1 s
3: www.google.com	5 s	5 s	2 s
4: www.baidu.com	-	-	5 s
5: weather API	-	-	10 s

Table 1 shows the system configurations of the three experiments. Client 1 asks Server 1 for the current time every 10 s during the five-hour sampling period. (The shell script executed in Client 1 is shown in Fig. 11 in Appendix A). The other clients generate background traffic. Clients 2–4 request Web pages, and Client 5 requests weather data in JavaScript Object Notation (JSON) format. The HTTP response payload length in Experiment 2 is about five times greater than that in Experiment 1. The number of packets generated by Clients 2–5 in Experiment 3 is about five times greater than those in Experiments 1 and 2.

Client 1 sends HTTP packets that are not used for XSS attacks. Performing the attacks is not necessary to evaluate our approach because our approach does not inspect contents in the HTTP frame and the travel and relay times are independent of XSS attacks. We examined the effect of the HTTP frame length by comparing the results of Experiments 1 and 2.

B. EVALUATION CRITERIA

The following procedure calculates five evaluation criteria from learning dataset $\mathcal{L}$ and evaluation dataset $\mathcal{E}$:

1) Calculate $\mathcal{L}_1$ and $\mathcal{E}_1$, which are the sets of "unique solutions" to Problem 1 derived from datasets $\mathcal{L}$ and $\mathcal{E}$, respectively. (As discussed later, not all unique solutions are included in $\mathcal{L}_1$).
2) Calculate 14 quantities (m_{req}, m_{res}, m_{12}, m_{34}, s_{req}, s_{res}, s_{12}, s_{34}, a_{res}, b_{res}, e_{res}, a_{req}, b_{req}, e_{req}) from solution set $\mathcal{L}_1$.
3) Calculate $\mathcal{E}_2$, which is the set of "solution sets" to Problem 2 derived from $\mathcal{E}$ and the above 14 quantities.
4) Calculate the five criteria defined by

$$R_c := \frac{|\{p_k \in \mathcal{E}_1 | u_k^2 = 1, p_k = q_k^1\}|}{|\mathcal{E}_1|}, \tag{18}$$

$$R_{c2} := \frac{|\{p_k \in \mathcal{E}_1 | u_k^2 \geq 2, \exists_{1 \leq j \leq u_k^2} p_k = q_k^j\}|}{|\mathcal{E}_1|}, \tag{19}$$

$$R_i := \frac{|\{p_k \in \mathcal{E}_1 | u_k^2 = 1, p_k \neq q_k^1\}|}{|\mathcal{E}_1|}, \tag{20}$$

$$R_{i2} := \frac{|\{p_k \in \mathcal{E}_1 | u_k^2 \geq 2, \forall_{1 \leq j \leq u_k^2} p_k \neq q_k^j\}|}{|\mathcal{E}_1|}, \tag{21}$$

$$R_0 := \frac{|\{p_k \in \mathcal{E}_1 | u_k^2 = 0\}|}{|\mathcal{E}_1|}, \tag{22}$$

where $p_k \in \mathcal{E}_1$ and $\{q_k^1, q_k^2, \ldots\} \in \mathcal{E}_2$ are the unique solution and the solution set for the k-th attack in $\mathcal{E}$, respectively, and $u_k^2 = |\{q_k^1, q_k^2, \ldots\}|$.

Because $\mathcal{E}_1$ is the set of correct solutions, R_c (R_{c2}) denotes the probability that $\mathcal{P}2$ outputs the correct solution (more than one solution among which the correct solution exists). R_i (R_{i2}) is the probability that $\mathcal{P}2$ outputs an incorrect solution (more than one solution among which the correct solution does not exist). R_0 is the probability that $\mathcal{P}2$ outputs no solutions. Note that

$$R_c + R_{c2} + R_i + R_{i2} + R_0 = 1. \tag{23}$$

Let us now define two error rates. The Type II error rate ($E2$) is defined as the probability that $\mathcal{P}2$ does not output a solution set that includes the correct solution, although attack packets (pkt1,pkt2,pkt3,pk4) exist in the bridge's log file (we use "correct solution" and "attack packets" interchangeably); therefore,

$$E2 = R_i + R_{i2} + R_0. \tag{24}$$

The type I error rate ($E1$) is defined as the probability that $\mathcal{P}2$ outputs a non-empty solution set, although the bridge's log file does not contain attack packets. Without executing $\mathcal{P}2$ one more time after removing all attack packets from dataset $\mathcal{E}$, $E1$ can be derived as

$$E1 = R_i + R_{i2} + R_{c2}, \tag{25}$$

because, from (18)–(22),

$$R_i^- + R_{i2}^- = R_i + R_{i2} + R_{c2}, \tag{26}$$

$$R_0^- = R_0 + R_c, \tag{27}$$

where R_i^-, R_{i2}^-, and R_0^- respectively denote R_i, R_{i2}, and R_0 after the removal of all attack packets.

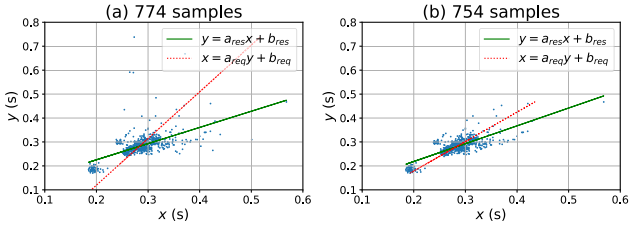

FIGURE 4. Scatter plots of (x_i, y_i) and least-squares lines obtained from (a) 774 samples and (b) 754 samples satisfying (28). For the 754 samples, $a_{req} = 0.74$, $b_{req} = 0.070$, $e_{req} = 0.024$, $a_{res} = 0.81$, $b_{res} = 0.055$, $e_{res} = 0.026$.

C. STATISTICS

Program $\mathcal{P}_1$ calculated 774 unique solutions from dataset $\mathcal{L}$ in Experiment 1. Let (x_i, y_i) be the i-th sample of (T_{req}, T_{res}) derived from the solutions. We define $\mathcal{L}_1$ as the set of all unique solutions satisfying

$$\frac{\max(x_i, y_i)}{\min(x_i, y_i)} < 1.3, \tag{28}$$

which requires that T_{req} and T_{res} are not widely different. Tor establishes a circuit between the client and server before data transmission; therefore, HTTP request and response messages use the same circuit. Fig. 4(a) shows the scatter plot of all 774 (x_i, y_i) samples and Fig. 4(b) shows that of 754 (x_i, y_i) samples that satisfy (28). As shown in the figures, (28) removes 20 samples (x_i, y_i) and their x_i or y_i values are considerably large. This phenomenon occurs because of TCP retransmissions. The figure also indicates that the least-squares regression lines are very sensitive to these outliers. The coefficient of determination R^2 indicates how well the observed outcomes are replicated by the regression model. After the removal, R^2_{res} (R^2_{req}), which denotes R^2 for the prediction of T_{res} (T_{req}), increases from 0.35 (0.32) to 0.61 (0.59).

During the five-hour sampling period, 42 circuits were established between Client 1 and Server 1, 31 of which were the main ones used. Thus, the circuit was replaced roughly every 10 m. Because onion routers are installed worldwide, travel times T_{req} and T_{res} depend on established circuits. Note that the entry router (i.e., the bridge) does not change even if the circuit changes. As shown in Fig. 4, there exist at least two circuit groups; one group has a travel time of around 0.2 s and the second one of around 0.3 s.

Fig. 5 shows histograms of the travel and relay times. It can be seen that the shapes of the T_{req} and T_{res} histograms are similar and that there exist a small number of T_{12} and T_{34} samples that are considerably larger or smaller than their means. These samples require a large α_3 value to meet Conditions 9 and 10 in Problem 2. Table 2 compares the statistics in the three experiments. It can be seen that the experimental conditions do not significantly affect the statistics, except for the standard deviations of packet relay times s_{12} and s_{34}.

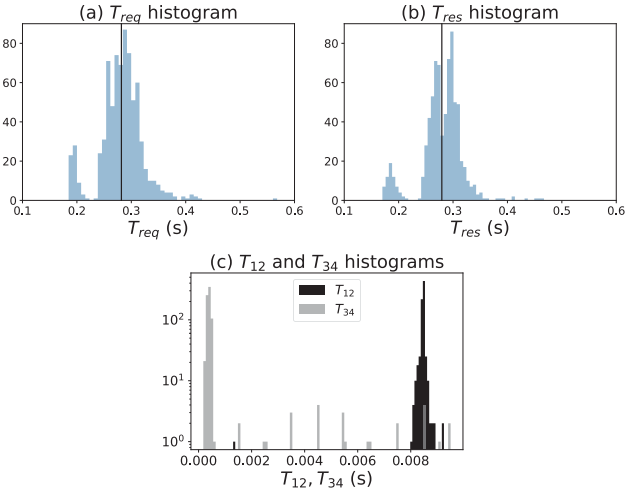

FIGURE 5. Histograms of (a) T_{req}, (b) T_{res}, (c) T_{12} and T_{34} derived from $|\mathcal{L}_1|(= 754)$ samples. $m_{req} = 0.28$, $s_{req} = 0.040$, $m_{res} = 0.28$, $s_{res} = 0.039$, $m_{12} = 8.5 \times 10^{-3}$, $s_{12} = 2.8 \times 10^{-4}$, $m_{34} = 5.8 \times 10^{-4}$, and $s_{34} = 1.1 \times 10^{-3}$ s.

TABLE 2. Statistics obtained in the three experiments (in seconds). Sample sizes $|\mathcal{L}_1|$ in Experiments 1, 2, and 3 were 754, 947, and 619, respectively.

	Exp. 1	Exp. 2	Exp. 3
m_{req}, s_{req}	0.28, 0.040	0.29, 0.087	0.27, 0.037
m_{res}, s_{res}	0.28, 0.039	0.28, 0.079	0.27, 0.036
m_{12}, s_{12}	0.0085, 0.00028	0.0099, 0.0256	0.0090, 0.0078
m_{34}, s_{34}	0.00058, 0.0011	0.00061, 0.001	0.0012, 0.0085
R^2_{res}, e_{res}	0.61, 0.024	0.88, 0.028	0.73, 0.019
R^2_{req}, e_{req}	0.59, 0.026	0.80, 0.037	0.72, 0.020

D. EXPERIMENTAL RESULTS

We consider the following four calculation options to understand the effectiveness of optional conditions 7–10 in Problem 2.

- Option 1. Conditions 1–6 are used.
- Option 2. Conditions 1–8 are used.
- Option 3. Conditions 1–6, 9, and 10 are used.
- Option 4. Conditions 1–10 are used.

In the calculation, the values of parameters α_1, α_2, and α_3 in program $\mathcal{P}2$ are the same as long as they are used. We introduce variable n to indicate that

$$n = \alpha_1 = \alpha_2 = \alpha_3. \tag{29}$$

Fig. 6 shows five evaluation criteria R_c, R_{c2}, R_i, R_{i2}, and R_0 as functions of n for all the experiment and option pairs. It can be seen that the values of R_i and R_{i2} are both small; therefore, $E2(=R_i+R_{i2}+R_0) \approx R_0$ and $E1(=R_i+R_{i2}+R_{c2}) \approx R_{c2}$. The figure also indicates that R_0 (R_{c2}) tends to decrease (increase) with n. (We explain the reason in Section VII-B). Thus, to minimize $\max(E1, E2)$, a value of n that satisfies $R_0 = R_{c2}$, which represents the intersection point of R_0 and R_{c2} curves, should be used. Let us define n^*_{err} as

$$\{n^*_{err}\} = \arg\min(\max(E1, E2)). \tag{30}$$

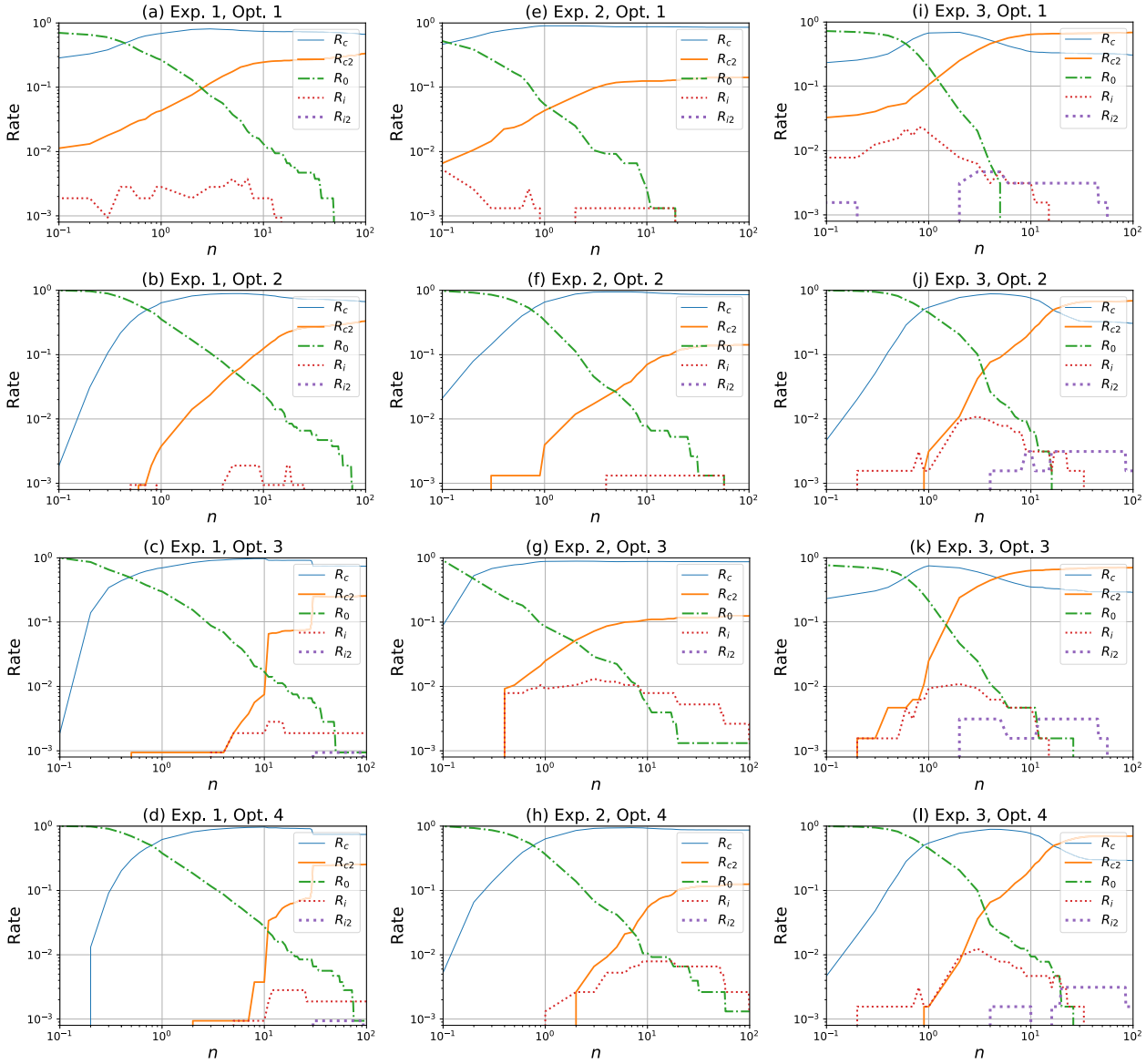

FIGURE 6. R_c, R_{c2}, R_i, R_{i2}, and R_0 for the three experiments and four options. The intersection point of R_0 and R_{c2} curves in each graph approximately agrees with the point where $\max(E1, E2)$ has the smallest value. $|\mathcal{E}_1|$ is equal to 1064, 760, and 641 for Experiments 1, 2, and 3, respectively. Both axes are log scale.

Furthermore, it is interesting that for all graphs in Fig. 6,

$$R_c^* \approx R_c(n_{err}^*), \tag{31}$$

where R_c^* and $R_c(n_{err}^*)$ are the maximum of R_c and R_c at $n = n_{err}^*$, respectively.

Table 3 shows the $E1$ and $E2$ rates for different n values. In the table, it can be seen that Option 4 at $n = [n_{err}^*]$ consistently provided small $E1$ and $E2$ values and large R_c values in all the experiments, where $[n_{err}^*]$ denotes integer n that minimizes $\max(E1, E2)$. Thus, a good strategy is to apply Option 4 and $n = [n_{err}^*]$, which we call the *min-max* strategy; it was subsequently used as a baseline. In Table 3, it can be seen that this strategy yielded $\max(E1, E2) \leq 7.6\%$ in all the experiments. However, the strategy is somewhat sensitive to the settings of Experiments 1–3 in that $[n_{err}^*]$ varies between 4 and 10 depending on the experiment (see Table 3 and Figs. 6(d), (h), and (l)).

An additional finding that can be observed in Fig. 6, which leads us to a different strategy development, is that Option 1 (Option 4) significantly reduces the value of R_0 (R_{c2}) at large (small) values of n. Table 3 numerically supports this observation. The table indicates that in all the experiments, Option 1 (Option 4) provided the widest range of n satisfying $E2 \leq \delta_i$ ($E1 \leq \delta_i$), $i = 1, 2$, where $\delta_1 = 1/641 (\lessapprox 0.0016)$ and $\delta_2 = 0.005$. (The reason is explained in Section VII-B). We should also mention that Experiment 3 yielded larger $E1$ and $E2$ values than the other experiments in most cases.

TABLE 3. $E1$, $E2$, and R_c (%) at $n = [n^*_{err}]$ and $E1$ ($E2$) (%) at the smallest (largest) n that satisfies $E2 \le \delta_i$ ($E1 \le \delta_i$), where $i = 1, 2$ and $\delta_1 = 1/641 (\lessapprox 0.16\%)$ and $\delta_2 = 0.5\%$. An n value attached to "$\ge$" ("$\le$") indicates the smallest (largest) n. "-" denotes there are no n values in (0,100] satisfying the constraint.

Exp.	Opt.	$\min\max(E1, E2)$			$E2 \le \delta_1$		$E2 \le \delta_2$		$E1 \le \delta_1$		$E1 \le \delta_2$	
		$[n^*_{err}]$	$E1, E2$	R_c	n	$E1$	n	$E1$	n	$E2$	n	$E2$
1	1	3	11.7, 7.7	80.8	≥ 49	29.0	≥ 22	26.3	-		-	
	2	5	5.4, 5.8	89.0	≥ 73	31.0	≥ 34	27.0	≤ 0.5	67.9	≤ 1.0	35.5
	3	10	0.9, 1.9	97.4	-		≥ 49	25.3	≤ 2.0	15.4	≤ 6.0	4.0
	4	10	0.5, 2.8	96.8	-		≥ 73	25.5	≤ 4.0	8.6	≤ 10	2.8
2	1	1	4.3, 5.4	90.3	≥ 20	13.2	≥ 10	12.6	-		-	
	2	5	2.9, 2.8	94.5	≥ 58	14.1	≥ 27	12.9	≤ 0.9	40.1	≤ 1.1	29.7
	3	2	5.4, 5.9	88.8	-		≥ 56	12.8	≤ 0.3	31.8	≤ 0.3	31.8
	4	7	2.9, 3.0	94.7	-		≥ 58	12.6	≤ 1.1	32.6	≤ 1.7	18.4
3	1	1	12.5, 21.8	67.6	≥ 57	67.4	≥ 11	65.4	-		-	
	2	4	8.7, 3.7	88.3	-		≥ 23	60.4	≤ 0.1	99.5	≤ 1.0	45.2
	3	1	3.4, 22.3	75.2	≥ 57	70.4	≥ 16	66.1	≤ 0.1	76.8	≤ 0.3	69.3
	4	4	7.6, 4.1	89.4	-		≥ 27	54.7	≤ 0.1	99.5	≤ 1.1	39.3

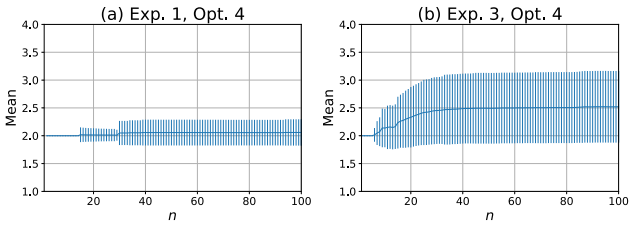

FIGURE 7. Means of $|\{u^2_k | u^2_k > 1, \exists_{1 \le j \le u^2_k} p_k = q^j_k\}|$. Error bars denote standard deviations. (a): Experiment 1 and Option 4. (b): Experiment 3 and Option 4.

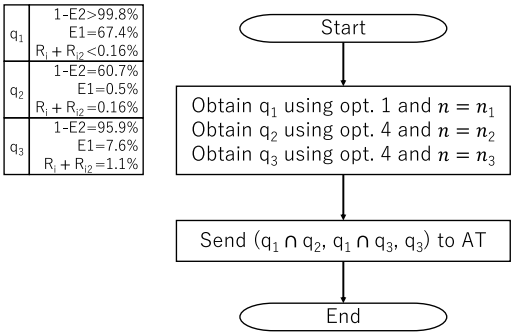

FIGURE 8. The procedure performed by cooperators, in which $n_1 = 57$, $n_2 = 1.1$, and $n_3 = 4$. The upper left-hand table shows the error rates in Experiment 3. "$1 - E2 > 99.8\%$" and "$R_i + R_{i2} < 0.16\%$" are equivalent to $R_i = R_{i2} = R_0 = 0$ and $R_i = R_{i2} = 0$, respectively.

Fig. 7 shows the mean sizes of the solution sets when the sets include the correct solution and their size is more than one. The figure demonstrates that the mean sizes are small. We found that, regardless of the options, Experiments 1 and 2 always showed means close to 2.0 and small standard deviations, whereas Experiment 3 yielded slightly larger means and standard deviations. These results imply that R_{c2} can be regarded as the rate of solution sets that contain one correct solution and only one or two incorrect solutions in most cases.

VII. DATA ANALYSIS

A. BOUNDS OF ERROR RATES

This section presents the idea of obtaining more than one solution set having different low error rates. The idea creates the notion of the reliability of solution sets and helps adequately grade the detection results obtained from many cooperators. Fig. 8 shows our approach for reducing the error rates. In the figure, q_1, q_2, and q_3 are solution sets obtained using Option 1 and $n = n_1$ satisfying

$$E2 = \delta_1, \tag{32}$$

Option 4 and $n = n_2$ satisfying

$$E1 = \delta_2, \tag{33}$$

and Option 4 and integer $n = n_3$ satisfying

$$\{n_3\} = \arg \min_{n \in \{1,2,\ldots\}} (\max(E1, E2)), \tag{34}$$

respectively. Table 3 shows that $n_1 = 57$, $n_2 = 1.1$, and $n_3 = 4$ for Experiment 3. We use these n_1, n_2, and n_3 values tuned to Experiment 3 because the detection in this case is the most difficult. Note that in Experiment 3

$$q_3 = q([n^*_{err}]), \tag{35}$$

where $q([n^*_{err}])$ is the solution set obtained by the min-max strategy.

Let us calculate the error rates of $q_1 \cap q_2$. Because q_1 (q_2) has a low $E2$ ($E1$) rate, the intersection of q_1 and q_2 is expected to render both the $E1$ and $E2$ rates low. To verify this, we first calculate the $E2$ rates. Assume that attack packets exist in the cooperator's log file. In this case, we introduce three events:

$$A_k := \{q_k \text{ has the correct solution}\}, \tag{36}$$

$$\bar{A}_k := \{q_k \text{ exists and does not have the correct solution}\}, \tag{37}$$

$$\bar{\bar{A}}_k := \{q_k \text{ does not have the correct solution}\}. \tag{38}$$

We use "q_k exists" and "$q_k \neq \emptyset$" interchangeably. Then,

$$P(A_k) = 1 - E2, \tag{39}$$

$$P(\bar{A}_k) = R_i + R_{i2}. \tag{40}$$

According to Appendix B, the probability that $q_1 \cap q_2$ causes a Type II error when $q_1 \cap q_2$ exists, i.e., the probability that all solutions in $q_1 \cap q_2$ are incorrect when it exists, satisfies

$$P(\bar{\bar{A}}_1 \cup \bar{\bar{A}}_2 | q_1 \cap q_2 \neq \emptyset) \leq \frac{P(\bar{A}_1) + P(\bar{A}_2)}{P(A_1) + P(A_2) - 1}. \quad (41)$$

The upper left-hand table in Fig. 8 shows the error rates in Experiment 3. According to the table and (39)–(40), (41) is given as

$$\frac{P(\bar{A}_1) + P(\bar{A}_2)}{P(A_1) + P(A_2) - 1} < \frac{0.0016 + 0.0016}{0.998 + 0.607 - 1} < 0.0052. \quad (42)$$

Similarly, the upper bound of the probability that $q_1 \cap q_3 (\neq \emptyset)$ causes a Type II error is

$$P(\bar{\bar{A}}_1 \cup \bar{\bar{A}}_3 | q_1 \cap q_3 \neq \emptyset) < 0.0131. \quad (43)$$

Thus, if $q_1 \cap q_2$ and $q_1 \cap q_3$ exist, they show very low Type II error rates. From (39), (60) in Appendix B, and the table in Fig. 8, the lower bounds of their existence probabilities are

$$P(q_1 \cap q_2 \neq \emptyset) \geq P(A_1) + P(A_2) - 1 > 0.605, \quad (44)$$
$$P(q_1 \cap q_3 \neq \emptyset) > 0.957. \quad (45)$$

We next consider the case where attack packets do not exist in the log file. In this case,

$$P(q_k \neq \emptyset) = E1. \quad (46)$$

From (46) and Fig. 8, the upper bounds of probabilities that $q_1 \cap q_2$ and $q_1 \cap q_3$ yield Type I errors are given by

$$P(q_1 \cap q_2 \neq \emptyset) \leq \min(P(q_1 \neq \emptyset), P(q_2 \neq \emptyset)) = 0.005, \quad (47)$$

$$P(q_1 \cap q_3 \neq \emptyset) \leq 0.076, \quad (48)$$

respectively.

TABLE 4. Upper bounds of Type I and II errors (%) of non-empty sets $q_1 \cap q_2$ and $q_1 \cap q_3$ (lower bounds of their existence rate (%) are in parentheses) and $E1$ and $E2$ (%) of q_3 and $q([n^*_{err}])$. The procedure in Fig. 8 calculates q_1, q_2, and q_3 with $n_1 = 57$, $n_2 = 1.1$, and $n_3 = 4$, respectively.

Exp.	Type	$q_1 \cap q_2$	$q_1 \cap q_3$	q_3	$q([n^*_{err}])$
1	I	0.09	0.09	0.09	0.5
	II	0.29 (65.4)	0.21 (91.3)	8.6	2.8
2	I	0.13	1.4	1.4	2.9
	II	0.39 (67.2)	0.70 (94.3)	5.5	3.0
3	I	0.5	7.6	7.6	7.6
	II	0.52 (60.5)	1.31 (95.7)	4.1	4.1

Table 4 summarizes the calculation results of (42)–(48). The table also shows the results of Experiments 1 and 2 that are derived using the same n_1, n_2, and n_3 values. These n values are tuned to Experiment 3, and thus, (32)–(35) do not hold in Experiments 1 and 2. For comparison, the error rates of $q([n^*_{err}])$ are included. Note that $q([n^*_{err}])$ is ideal in that the best n value is selected for each experiment. The table demonstrates the following:

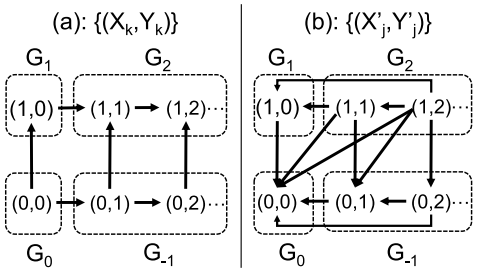

FIGURE 9. State transition diagrams of two Markov chains and lumped states G_0, G_1, G_2, and G_{-1}. (a): $\{(X_k, Y_k)\}$. (b): $\{(X'_j, Y'_j)\}$.

- $q_1 \cap q_2 (\neq \emptyset)$ is the most reliable solution. Its Type I (II) error rate is less than one-fifth (one-seventh) of that of $q([n^*_{err}])$ for all the experiments. Its existence bounds (60.5–67.2%), however, are not large.
- $q_1 \cap q_3 (\neq \emptyset)$ is slightly less reliable (it is still superior to $q([n^*_{err}])$) and its existence bounds are 91.3–95.7%.
- If $q_1 \cap q_2 = q_1 \cap q_3 = \emptyset$, the min-max strategy $q([n^*_{err}])$ should be used (therefore, its approximation q_3 is sent to AT, as shown in Fig. 8).

Let us define the reliability degree, for example, by the maximum of the upper bounds of Type I and II error rates. Smart contract AT receives `TX-detect` transactions including $(q_1 \cap q_2,\ q_1 \cap q_3,\ q_3)$ and their reliability degrees from cooperators, and ranks the solution sets based on the degrees. Note that different cooperators yield different reliability degrees of $q_1 \cap q_2 (\neq \emptyset)$.

B. MARKOVIAN PERSPECTIVE

Let us discuss the condition that the values of $E1$ and $E2$ become small. Regardless of whether attack packets exist in the cooperator's log file, each solution q is associated with a unique n value, n^+, such that program $\mathcal{P}2$ outputs a solution set containing q if $n \geq n^+$; otherwise it does not. Let n^+_k be the k-th smallest n^+; i.e.,

$$0 < n^+_1 < n^+_2 < \cdots. \quad (49)$$

Let X_k (Y_k) be the number of correct (incorrect) solutions at $n = n^+_k$. We can constitute a two-dimensional discrete Markov chain $\{(X_k, Y_k)\}_{k \geq 0}$ such that $X_0 = Y_0 = 0$ and for $k \geq 1$,

$$X_k = X_{k-1} + C_k, \quad (50)$$
$$Y_k = Y_{k-1} + I_k, \quad (51)$$

where $C_k = 1$ and $I_k = 0$ if the correct solution is associated with n^+_k; otherwise, $C_k = 0$ and $I_k = 1$. Fig. 9 illustrates the state transition diagram of $\{(X_k, Y_k)\}$ and lumped states defined by $G_0 := \{(0,0)\}$, $G_1 := \{(1,0)\}$, $G_2 := \{(r,s) | r = 1, s \geq 1\}$, and $G_{-1} := \{(r,s) | r = 0, s \geq 1\}$. Then, from (18)–(22),

$$P((X_k, Y_k) \in G_0) \sim R_0, \quad (52)$$
$$P((X_k, Y_k) \in G_1) \sim R_c, \quad (53)$$

$$P((X_k, Y_k) \in G_2) \sim R_{c2}, \tag{54}$$

$$P((X_k, Y_k) \in G_{-1}) \sim R_i + R_{i2}, \tag{55}$$

where $\beta \sim b$ denotes b is an estimator of β.

In Fig. 9(a), G_2 (G_0) has only incoming (outgoing) links. This finding agrees with the fact that the value of R_{c2} (R_0) does not decrease (increase) with the value of n in all the graphs in Fig. 6. According to Fig. 6, R_0 somewhat linearly decreases in the range of $n > 10^0$ on a log-log scale; i.e., $R_0 = c_1\, n^{c_2}$, $c_1 > 0$ and $c_2 < 0$, which represents a power-law function. Although the value of R_0 may not quickly decrease, if Option 1 is used and attack packets exist in the log file, we eventually have $E2 = 0$ at a large value of k because there exist n_k^+ values at which $T_1' \leq T_1$ and $T_4 \leq T_4'$ hold.

Let us next define another Markov chain. We introduce a new parameter α_j each time new optional conditions are added to Problem 2 and revise (29) as

$$n = \alpha_1 = \alpha_2 = \cdots = \alpha_j. \tag{56}$$

Then, we obtain Markov chain $\{(X_j', Y_j')\}_{j\geq 1}$, where X_j' (Y_j') denotes the number of correct (incorrect) solutions when j parameters $\alpha_1, \alpha_2, \ldots, \alpha_j$ are used. Fig. 9(b) shows the state transition diagram of $\{(X_j', Y_j')\}$, which is derived from the principle that X_j' and Y_j' do not increase with the number of constraints j. In the figure, G_2 (G_0) has only outgoing (incoming) links. This result explains why the value of R_{c2} (R_0) of Option 4 is not greater (smaller) than that of any other options at any n value in Fig. 6.

In short, two chains $\{(X_k, Y_k)\}$ and $\{(X_j', Y_j')\}$ indicate

$$E1 \uparrow E2 \downarrow \text{ as } k \uparrow \text{ or } j \downarrow \tag{57}$$

if the value of $R_i + R_{i2}$ is sufficiently small. Based on the relationship represented by (57), Fig. 8 shows the calculation of q_1, using Option 1 and a large n value to reduce $E2$, and q_2, using Option 4 and a small n value to reduce $E1$. The intersection of q_1 and q_2 is then taken to make $E1$ and $E2$ both small.

VIII. DISCUSSION

In this paper, we proposed a packet detection approach, in which an entry router's log file is analyzed to determine the packets that correspond to a request and response pair observed on the server. The application scope of this approach may be enlarged and its detection accuracy improved by adding, deleting, or changing the conditions given in Problems 1 and 2. If the detection of only an HTTP request message is required, the approach can be adapted to this requirement by deleting all the conditions related to packets pkt3 and pkt4 in the problems. If the HTTP response consists of multiple IP packets, the approach is effective after conditions that select one packet, e.g., the first packet, from the packet group are added. This is rendered possible by checking the TCP sequence numbers. Furthermore, according to (57), the Type I error rate may be further reduced by adding more conditions to Problem 2.

Our approach may not require many onion routers to be cooperators. This is because the directory servers can provide information to clients such that the clients select cooperators as their entry routers with a high probability.

IX. CONCLUSIONS

We have thus far investigated means of identifying attackers' packets that may be recorded in an onion router's log file with sufficiently low error rates and without compromising Tor users' privacy. Our approach was to minimize the detection error rates rather than to maximize the detection success rate. Thus, our approach may not output a single candidate. The ideas presented in this paper were as follows. (1) Attacked servers and cooperators, i.e., routers that voluntarily agree to trace attackers, form an Ethereum network, in which open and tamper-proof blockchain technologies prevent the counterfeiting of evidence files and allow Ethereum participants to monitor the tracing processes of all incidents in their entirety. (2) Before performing attack-packet detection, cooperators collect travel and relay time samples with the support of the attacked server. (3) Each detection result includes a reliability degree that is based on its error rates, and thus, smart contract AT can rank the detection results reported by cooperators.

We performed three different experiments using a bridge (a private entry router), which detects attack packets that have been mixed with normal packets. We learned the following by analyzing the detection results: (1) The min-max strategy, which minimizes the maximum of Type I and II errors, achieves error rates of not more than 7.6%. (2) The Type I and II errors of a non-empty solution set obtained by our approach respectively become less than one-fifth and one-seventh of those yielded by the min-max strategy. This error-rate reduction approach is supported by two Markov chain models introduced in this paper.

APPENDIX A
LOG FILES

Fig. 10 shows the tcpdump log data of the bridge and the server. As shown in Fig. 10(a), there are two packets in the bridge's log file, and `TS var` in the first packet is equal to `TS ecr` in the second. Therefore, they may be an HTTP request and response pair. Meanwhile, the server's log file includes unencrypted packets, and thus, we can check whether the HTTP request and response messages have the same sequence number `torclientid`. The messages in Figs. 10(b) and (c) have the same `torclientid`.

Fig. 11 shows the shell script on the client and a part of its output data. As shown in Fig. 11(a), the script requests the current time from the server every 10 s, and each request has a unique sequence number (`torclientid`). The output data in Fig. 11(b) are used for four purposes.

1) The HTTP status code in Fig. 11(b) is checked whether it is 200.
2) `endtime` in Fig. 11(b) is used as T_4 in Fig. 1.
3) Sequence number `torclientid` in Fig. 11(b) is compared with those in HTTP messages in the server's

(a)
```
2019-12-09 20:00:00.587625 IP 150.43.248.246.41448 > 176.9.40.131.443: Flags [P.], seq
44170:44713, ack 223258, win 1444, options [nop,nop,TS val 4246764946 ecr 191029952],
length 543
2019-12-09 20:00:00.939454 IP 176.9.40.131.443 > 150.43.248.246.41448: Flags [P.], seq
223258:223801, ack 44713, win 999, options [nop,nop,TS val 191030074 ecr 4246764946],
length 543
```

(b)
```
2019-12-09 20:00:04.683088 IP 167.88.7.134.42695 > 150.43.248.251.80: Flags [P.], seq
3068490940:3068491050, ack 1569383879, win 1026, options [nop,nop,TS val 1770049985
ecr 2929116830], length 110: HTTP: GET /getdate.php?torclientid=242.1868 HTTP/1.1
        0x0000:  4500 00a2 0000 4000 2b06 1151 a758 0786  E.....@.+..Q.X..
        0x0010:  962b f8fb a6c7 0050 b6e5 74bc 5d8a e5c7  .+.....P..t.]...
        0x0020:  8018 0402 c171 0000 0101 080a 6980 d1c1  .....q......i...
        0x0030:  ae96 c69e 4745 5420 2f67 6574 6461 7465  ....GET./getdate
        0x0040:  2e70 6870 3f74 6f72 636c 6965 6e74 6964  .php?torclientid
        0x0050:  3d32 3432 2e31 3836 3820 4854 5450 2f31  =242.1868.HTTP/1
        0x0060:  2e31 0d0a 486f 7374 3a20 3135 302e 3433  .1..Host:.150.43
        0x0070:  2e32 3438 2e32 3531 0d0a 5573 6572 2d41  .248.251..User-A
        0x0080:  6765 6e74 3a20 6375 726c 2f37 2e34 372e  gent:.curl/7.47.
        0x0090:  300d 0a41 6363 6570 743a 202a 2f2a 0d0a  0..Accept:.*/*..
        0x00a0:  0d0a
```

(c)
```
2019-12-09 20:00:04.684112 IP 150.43.248.251.80 > 167.88.7.134.42695: Flags [P.], seq
1:266, ack 110, win 227, options [nop,nop,TS val 2929117399 ecr 1770049985], length 265:
HTTP: HTTP/1.1 200 OK
        0x0000:  4500 013d 004f 4000 4006 fb66 962b f8fb  E..=.O@.@..f.+..
        0x0010:  a758 0786 0050 a6c7 5d8a e5c7 b6e5 752a  .X...P..]....u*
        0x0020:  8018 00e3 2117 0000 0101 080a ae96 c8d7  ....!...........
        0x0030:  6980 d1c1 4854 5450 2f31 2e31 2032 3030  i...HTTP/1.1.200
        0x0040:  204f 4b0d 0a44 6174 653a 204d 6f6e 2c20  .OK..Date:.Mon,.
        0x0050:  3039 2044 6563 2032 3031 3920 3131 3a30  09.Dec.2019.11:0
        0x0060:  303a 3034 2047 4d54 0d0a 5365 7276 6572  0:04.GMT..Server
        0x0070:  3a20 4170 6163 6865 2f32 2e34 2e31 3820  :.Apache/2.4.18.
        0x0080:  2855 6275 6e74 7529 0d0a 5661 7279 3a20  (Ubuntu)..Vary:.
        0x0090:  4163 6365 7074 2d45 6e63 6f64 696e 670d  Accept-Encoding.
        0x00a0:  0a43 6f6e 7465 6e74 2d4c 656e 6774 683a  .Content-Length:
        0x00b0:  2039 340d 0a43 6f6e 7465 6e74 2d54 7970  .94..Content-Typ
        0x00c0:  653a 2074 6578 742f 6874 6d6c 3b20 6368  e:.text/html;.ch
        0x00d0:  6172 7365 743d 5554 462d 380d 0a0d 0a74  arset=UTF-8....t
        0x00e0:  696d 6520 3d20 3135 3735 3838 3932 3034  ime.=.1575889204
        0x00f0:  2e36 3833 3620 3c70 3e32 3031 392d 3132  .6836.<p>2019-12
        0x0100:  2d30 3920 3131 3a30 303a 3034 3c70 3e31  -09.11:00:04<p>1
        0x0110:  3637 2e38 382e 372e 3133 343c 703e 746f  67.88.7.134<p>to
        0x0120:  7263 6c69 656e 7469 643d 3234 322e 3138  rclientid=242.18
        0x0130:  3638 3c70 3e35 3030 3430 3738 34              68<p>50040784
```

FIGURE 10. Tcpdump log data in the bridge (a) and in the server ((b) and (c)). (a): TS val and ecr connect two packets. (b): Each HTTP request has sequence number `torclientid`. (c): Each HTTP response has `torclientid` and current time `time`.

(a)
```
#!/bin/bash
i=0
while :
do
    echo "begintime="`date +%s.%6N` >> getdate242.txt
    torify curl -X GET -i 150.43.248.251/getdate.php?torclientid=242."$i" >> getdate242.txt
    echo -e >> getdate242.txt
    echo -e "endtime="`date +%s.%6N` >> getdate242.txt
    echo -e "=======" >> getdate242.txt
    /bin/sleep 10
    let "i++"
done
```

(b)
```
begintime=1575889203.830478
HTTP/1.1 200 OK
Date: Mon, 09 Dec 2019 11:00:04 GMT
Server: Apache/2.4.18 (Ubuntu)
Vary: Accept-Encoding
Content-Length: 94
Content-Type: text/html; charset=UTF-8

time = 1575889204.6836 <p>2019-12-09 11:00:04<p>167.88.7.134<p>torclientid=242.1868
<p>50040784
endtime1575889204.967275
======
```

FIGURE 11. (a): Shell script executed on the client. (b): Part of the output data (getdate242.txt) of the shell script, which include sequence number `torclientid` and current time `time` inserted in the response.

log file. In Figs. 10(b), 10(c), and 11(b), they are the same.

4) Current time `time` in Fig. 11(b) is used to detect an HTTP response that has the same `time`. Figs. 10(c) and 11(b) illustrate the case where they are the same.

APPENDIX B
PROBABILITY CALCULATION

First,

$$\{A_1 \cap A_2\} \subset \{q_1 \cap q_2 \neq \emptyset\}, \tag{58}$$

$$\{q_1 \cap q_2 \neq \emptyset\} = \{q_1 \neq \emptyset\} \cap \{q_1 \cap q_2 \neq \emptyset\}. \tag{59}$$

Accordingly,

$$P(q_1 \cap q_2 \neq \emptyset) \geq P(A_1 \cap A_2) \geq P(A_1) + P(A_2) - 1, \tag{60}$$

$$\bar{\bar{A}}_1 \cap \{q_1 \cap q_2 \neq \emptyset\} = \bar{\bar{A}}_1 \cap \{q_1 \neq \emptyset\} \cap \{q_1 \cap q_2 \neq \emptyset\} \tag{61}$$

$$= \bar{A}_1 \cap \{q_1 \cap q_2 \neq \emptyset\}. \tag{62}$$

Using (60)–(62), we have

$$P(\bar{\bar{A}}_1 \cup \bar{\bar{A}}_2 | q_1 \cap q_2 \neq \emptyset)$$

$$= \frac{P((\bar{\bar{A}}_1 \cup \bar{\bar{A}}_2) \cap \{q_1 \cap q_2 \neq \emptyset\})}{P(q_1 \cap q_2 \neq \emptyset)} \tag{63}$$

$$= \frac{P((\bar{A}_1 \cup \bar{A}_2) \cap \{q_1 \cap q_2 \neq \emptyset\})}{P(q_1 \cap q_2 \neq \emptyset)} \leq \frac{P(\bar{A}_1 \cup \bar{A}_2)}{P(q_1 \cap q_2 \neq \emptyset)} \tag{64}$$

$$\leq \frac{P(\bar{A}_1) + P(\bar{A}_2)}{P(A_1) + P(A_2) - 1}. \tag{65}$$

REFERENCES

[1] R. Dingledine, N. Mathewson, S. Murdoch, and P. Syverson, "Tor: The second-generation onion router (2014 DRAFT v1)," *Cl. Cam. Ac. Uk*, 2014.
[2] T. T. Project. (2020). *Tor Metrics*. [Online]. Available: https://metrics.torproject.org/
[3] M. Casenove and A. Miraglia, "Botnet over tor: The illusion of hiding," in *Proc. 6th Int. Conf. Cyber Conflict*, Jun. 2014, pp. 273–282.
[4] A. Sanatinia and G. Noubir, "OnionBots: Subverting privacy infrastructure for cyber attacks," in *Proc. 45th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Jun. 2015, pp. 69–80.
[5] N. Christin, "Traveling the silk road: A measurement analysis of a large anonymous online marketplace," in *Proc. 22nd Int. Conf. World Wide Web*, 2013, pp. 213–224.
[6] D. Gonzalez and T. Hayajneh, "Detection and prevention of crypto-ransomware," in *Proc. IEEE 8th Annu. Ubiquitous Comput., Electron. Mobile Commun. Conf. (UEMCON)*, Oct. 2017, pp. 472–478.
[7] A. Sanatinia and G. Noubir, "Honions: Towards detection and identification of misbehaving tor hsdirs," in *Proc. Workshop Hot Topics Privacy Enhancing Technol. (HotPETs)*, 2016, pp. 1–2.
[8] N. Hopper, "Protecting Tor from Botnet abuse in the long term," The Tor Project, Seattle, WA, USA, Tech. Rep. 2013–11-001, 2013.
[9] Z. Ling, J. Luo, K. Wu, W. Yu, and X. Fu, "Torward: Discovery, blocking, and traceback of malicious traffic over tor," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 12, pp. 2515–2530, 2015.
[10] Z. Liu, Y. Liu, P. Winter, P. Mittal, and Y.-C. Hu, "TorPolice: Towards enforcing service-defined access policies for anonymous communication in the tor network," in *Proc. IEEE 25th Int. Conf. Netw. Protocols (ICNP)*, Oct. 2017, pp. 1–10.
[11] Z. Yao, J. Ge, Y. Wu, X. Zhang, Q. Li, L. Zhang, and Z. Zou, "Meek-based tor traffic identification with hidden Markov model," in *Proc. IEEE 20th Int. Conf. High Perform. Comput. Commun.*, Jun. 2018, pp. 335–340.
[12] B. Cusack, Z. Tian, and A. K. Kyaw, "Identifying dos and ddos attack origin: Ip traceback methods comparison and evaluation for iot," in *Interoperability, Safety and Security in IoT*. Cham, Switzerland: Springer, 2016, pp. 127–138.
[13] S. Saleh, J. Qadir, and M. U. Ilyas, "Shedding light on the dark corners of the Internet: A survey of tor research," *J. Netw. Comput. Appl.*, vol. 114, pp. 1–28, Jul. 2018.
[14] V. Buterin, "Ethereum: A next-generation cryptocurrency and decentralized application platform," *Bitcoin. Mag.*, vol. 23, p. 252, Oct. 2014.
[15] A. Sanatinia, "Abusing privacy infrastructures: Case study of tor," Ph.D. dissertation, College Comput. Inf. Sci., Northeastern Univ., Boston, MA, USA, 2018.
[16] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Manubot, Tech. Rep., 2019. [Online]. Available: https://git.dhimmel.com/bitcoin-whitepaper/
[17] A. Kwon, M. AlSabah, D. Lazar, M. Dacier, and S. Devadas, "Circuit fingerprinting attacks: Passive deanonymization of tor hidden services," in *Proc. 24th Secur. Symp.*, 2015, pp. 287–302.
[18] G. Cherubin, J. Hayes, and M. Juarez, "Website fingerprinting defenses at the application layer," *Proc. Privacy Enhancing Technol.*, vol. 2017, no. 2, pp. 186–203, Apr. 2017.

[19] L. Overlier and P. Syverson, "Locating hidden servers," in *Proc. IEEE Symp. Secur. Privacy*, 2006, p. 15.
[20] L. Zhang, J. Luo, M. Yang, and G. He, "Application-level attack against Tor's hidden service," in *Proc. 6th Int. Conf. Pervas. Comput. Appl.*, Oct. 2011, pp. 509–516.
[21] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson, "Users get routed: Traffic correlation on tor by realistic adversaries," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2013, pp. 337–348.
[22] K. Müller, "Defending end-to-end confirmation attacks against the tor network," M.S. thesis, Dept. Comput. Sci., MTDMT Technol., Gjøvik Univ. College, Gjøvik, Norway, 2015.
[23] A. Chaabane, P. Manils, and M. A. Kaafar, "Digging into anonymous traffic: A deep analysis of the tor anonymizing network," in *Proc. 4th Int. Conf. Netw. Syst. Secur.*, Sep. 2010, pp. 167–174.
[24] V. Jacobson, B. Braden, and D. Borman, "TCP extensions for high performance," Internet Requests Comments, Tech. Rep. RFC 1323, May 1992. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1323.txt
[25] R. K. Chang, "Defending against flooding-based distributed denial-of-service attacks: A tutorial," *IEEE Commun. Mag.*, vol. 40, no. 10, pp. 42–51, Mar. 2002.
[26] A. Belenky and N. Ansari, "On IP traceback," *IEEE Commun. Mag.*, vol. 41, no. 7, pp. 142–153, Jul. 2003.
[27] K. Singh, P. Singh, and K. Kumar, "A systematic review of IP traceback schemes for denial of service attacks," *Comput. Secur.*, vol. 56, pp. 111–139, Feb. 2016.
[28] H. Burch and B. Cheswick, "Tracing anonymous packets to their approximate source," in *Proc. LISA*, 2000, pp. 319–327.
[29] S. M. Bellovin, M. Leech, and T. Taylor, "ICMP traceback messages," Internet Eng. Task Force, Fremont, CA, USA, Tech. Rep., 2003.
[30] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Network support for IP traceback," *IEEE/ACM Trans. Netw.*, vol. 9, no. 3, pp. 226–237, Jun. 2001.
[31] A. C. Snoeren, "Hash-based IP traceback," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 4, pp. 3–14, Oct. 2001.
[32] R. Stone, "Centertrack: An ip overlay network for tracking dos floods," in *Proc. USENIX Secur. Symp.*, vol. 21, 2000, p. 114.
[33] Q. Xiaofeng, H. Jihong, and C. Ming, "A mechanism to defend SYN flooding attack based on network measurement system," in *Proc. ITRE. 2nd Int. Conf. Inf. Technology: Res. Edu.*, 2004, pp. 208–212.
[34] *Txtorcon*. Accessed: Mar. 2020. [Online]. Available: https://txtorcon.readthedocs.io/en/latest/
[35] R. Dingledine and N. Mathewson, "Tor protocol specification," Tech. Rep., 2008. [Online]. Available: https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt
[36] C.-H. J. Wu and J. D. Irwin, *Introduction to Computer Networks and Cybersecurity*. Boca Raton, FL, USA: CRC Press, 2016.
[37] I. R. Learmonth and K. Loesing, "Tor metrics data collection, aggregation, and presentation," Tech. Rep. 2019-03-001, 2019.
[38] D. L. Mills, "Network time protocol (NTP)," Internet Requests Comments, Tech. Rep. RFC 958, Sep. 1985. [Online]. Available: http://www.rfc-editor.org/rfc/rfc958.txt
[39] V. Jacobson, C. Leres, and S. McCanne. (2003). *TCP Dump Public Repository*. [Online]. Available: http://www.tcpdump.org

YINAN PEI was born in Jinzhou, Liaoning, China, in 1992. He graduated from the Dalian University of Technology, in 2015. He received the master's degree from the Graduate School of Course of Computer Science and Engineering, Fukuoka Institute of Technology, in 2020. He is currently working with Nexis, as a Software Engineer. His interests include cryptography, security systems, and blockchain technologies.

KAZUMASA OIDA (Member, IEEE) received the bachelor's degree in information science from the University of Tsukuba, in 1983, the master's degree in engineering from Hokkaido University, in 1985, and the Ph.D. degree in informatics from Kyoto University, in 2002. He worked for the Nippon Telegraph and Telephone Corporation for 20 years, as an Engineer, where he has participated in the development of private network systems. He is currently a Professor with the Department of Computer Science and Engineering, Fukuoka Institute of Technology, Japan. His main interests include cybersecurity, social network analysis, and blockchain.