A De-anonymization Attack against Downloaders in Freenet

Yonghuan Xu[†], Ming Yang[‡*], Zhen Ling[‡], Zixia Liu[§], Xiaodan Gu[‡], Lan Luo[§]
[†]School of Cyber Science and Engineering, Southeast University, China
[‡]School of Computer Science and Engineering, Southeast University, China
Email: {xuyonghuan, yangming2002, zhenling, xdgu}@seu.edu.cn
[§]School of Computer Science and Technology, Anhui University of Technology, China
Email: {zxliu, lluo}@ahut.edu.cn

***Abstract*—Freenet is a well-known anonymous communication system that enables file sharing among users. It employs a probabilistic hops-to-live (HTL) decrement approach to hide the originator among nodes in a multi-hop path. Therefore, all nodes shall exhibit identical behaviors to preserve anonymity. However, we discover that the path folding mechanism in Freenet violates this principle due to behavior discrepancy between downloaders and intermediate nodes. The path folding mechanism is designed to optimize the network topology of Freenet. A delayed path folding message by a successor node may incur a timeout event at its predecessor, and an intermediate node reacts differently to such timeout with a downloader. Therefore, malicious nodes can deliberately trigger the timeout event to identify downloaders. The complex implementation of the path folding timeout detection mechanism in Freenet complicates our de-anonymization attack. We thoroughly analyze the underlying cause and develop three strategies to manipulate three types of messages respectively at the malicious node, minimizing the false positive rate. We conduct extensive real-world experiments to verify the feasibility and effectiveness of our attack. They show that our attack achieves a true positive rate of 100% and false positive rate of near 0% under two different Freenet download modes.**

I. Introduction

Freenet[1] [1], [2], [3] is a popular anonymous communication system that supports various applications such as file sharing and web forum. It is basically a peer-to-peer overlay network functioning as a distributed hash table (DHT) [4]. Users can upload files to or download files from the network anonymously through deploying a Freenet node. Each file in Freenet is segmented into encrypted blocks of fixed size and randomly distributed among Freenet nodes. Each Freenet node contributes local storage space and bandwidth resources for storing and transferring file blocks, respectively. The network is organized into a small-world network [5] to guarantee each node can reach rest nodes via a few hops. Freenet performs hop-by-hop routing and employs a probabilistic *hops-to-live* (HTL) decrement mechanism to enable file uploads and downloads through a multi-hop path and protect the anonymity for both the uploader and downloader by preventing the successor node in the path from identifying them.

Previous works have investigated the anonymity provided by Freenet. Tian *et al.* [6], [7] discover the information leakage caused by the loop detection mechanism during the routing process, and propose a traceback attack against downloaders by exploiting the loop detection protocol to uncover the routing path and identify the originating node. Levine *et al.*

* Corresponding author: Prof. Ming Yang of Southeast University, China.

[1]Freenet project has been renamed to Hyphanet, however, we keep using the original name since it is more familiar to the community.

interest even if the attacker forces a timeout. We design three different enhancement schemes by actively manipulating distinct critical messages sent to the intermediate node so that we can considerably reduce the false positive rate of our attack.

Our major contributions are summarized as follows.

- We are the first to expose fundamental weaknesses in Freenet anonymity protection from a protocol design standpoint. Specifically, upon thoroughly examining the Freenet protocol design, we discover an exploitable vulnerability in the path folding protocol for conducting de-anonymization attacks, enabling the identification of whether a victim predecessor node is a downloader.
- Freenet has a complex implementation of the path folding timeout detection mechanism and may not detect the path folding timeout correctly, leading to a significant increase in false positive rates in our attack. We carefully analyze the root cause and then design three schemes to manipulate three types of Freenet messages respectively, effectively addressing the imprecise timeout detection issue and reducing the false positive rate.
- We demonstrate the feasibility and effectiveness of our de-anonymization attack through extensive real-world experiments. The results reveal that the anonymity of downloaders in Freenet is severely undermined and a Freenet downloader node with default output bandwidth limit or less can be de-anonymized with a 100% true positive rate and a false positive rate close to 0%.

Responsible disclosure: We disclosed our de-anonymization attack to Freenet, which confirmed the vulnerability. We had been working with Freenet to test their patch until the vulnerability was completely mitigated [11].

II. Background on Freenet

In this section, we provide an overview of the Freenet network structure, the process of file uploading and downloading, as well as the path folding mechanism in Freenet.

A. Freenet network structure

Freenet is a peer-to-peer anonymous information storage and retrieval system, with nodes contributing their local storage space and bandwidth resources to the network. Each Freenet node is assigned a random location between 0 and 1 on a logical ring with a circumference of 1. A new node joins Freenet by firstly connecting with some *seed* nodes to discover other existing nodes, and then selecting some of the existing nodes as its peers (or neighbors). The number of peers a node can have is determined by the contributed output bandwidth. In Freenet, the distance between a node and 70% of its peers is within 0.01, while the remaining peers are more than 0.01 away from the node. This results in the formation of a small-world network in Freenet. A node can upload or download data stored in the Freenet network through its connected peers.

B. File uploading and downloading

In Freenet, each file has an associated uniform resource identifier (URI) with two options: content hash key (CHK) or signed subspace key (SSK). File uploading starts from chunking the file into encrypted 32KB file blocks and then generating a CHK for each block as the block URI. The CHK contains information such as the hash of the block, a decryption key, etc., where the hash of the block is also referred to as the *routing key* used to locate the block in Freenet. Then the CHKs of all blocks are enveloped into a single metadata file. However, if the metadata file is larger than 32KB, it can be further segmented into 32KB metadata blocks. The aforementioned block generation procedure will repeat until the final metadata size is no more than 32KB. If SSK is chosen as the file URI, the overall generation procedure is similar to that using CHK except the final metadata is limited to a maximum of 1KB, instead of 32KB. The CHK or SSK of the final metadata file serves as the URI for the entire file.

Each file block can be addressed by mapping its *routing key* to a location on the logic ring. This allows the file uploader (upstream originator) to select one of its peers closest to the block's location based on the locations of the peer and the peer's peers. The uploader then sends a request to the selected peer containing a unique identifier (UID), a value of hops-to-live (HTL), and the block *routing key*. The HTL (initially 18) normally decrease 1 at each hop until it reaches 0, preventing an endless routing request. Upon receiving the request, the selected downstream peer repeats peer selection process to forward the request. In this way, a routing path which consists of several nodes is created. An adjacent downstream peer in the routing path of a node is referred to as the node's *successor*, while an adjacent upstream peer as the node's *predecessor*. To prevent exposing the uploader from the HTL, the HTL decreases by a 50% probability at each hop when it is 18. The use of multi-hop routing and a probabilistic decrease of the initial HTL makes it difficult to identify the uploader. Once the HTL reaches a threshold of 16, an intermediate node in the path stores the block if it is closer to the block's location on the logic ring than the predecessor. After the entire file is uploaded, the file owner can share the file URI via an out-of-band channel, e.g., a public forum.

To download a file, the user first uses the file URI to locate the metadata that inclpeatses public f2-345.00028 (locle,)-309.9993seblo3

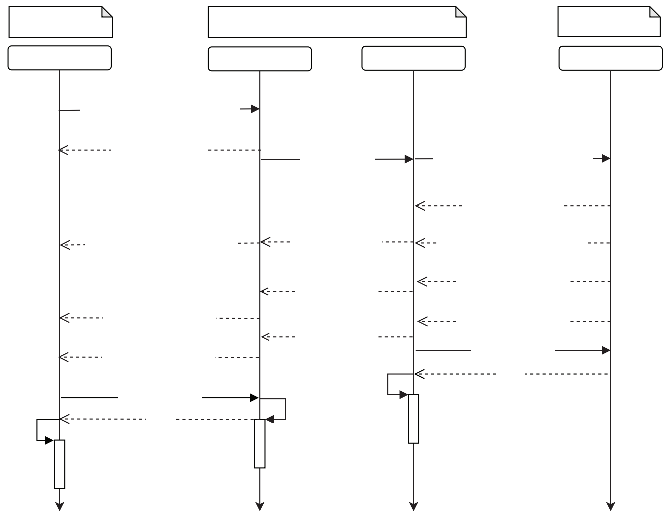


Fig. 1: Workflow of downloading a CHK block.

The path folding mechanism is performed on the path where a CHK block is successfully retrieved. Once the last hop node in the path completes transmitting a file block to its predecessor, it can send a path folding request message, followed by a series of messages conveying the *node reference*, to its predecessor based on its demand, i.e., it lacks a peer or needs replacing one of its peers. The *node reference* consists of various information about the node, such as its IP address, port, and location. Otherwise, it sends a termination message.

Once the message reaches the upstream peer, if it is a termination message, this node either re-initiates a new path folding request or send a termination message to its own predecessor based on its own demand, in the same way as the downstream node. If the message is a request, and this upstream node is interested (i.e. lacks a peer or needs replacing a peer), then this upstream node sends a bunch of response messages back via the path, which contain the *node reference* of itself. After receiving the response, the node which sends the request can directly establish a connection with it. If the message is a request but this upstream node is not interested, It either relays the request upstream with a probability of 95% or blocks the request with a 5% chance by sending a termination message to its successor, and then determines whether to initiate a path folding request upstream. In the case when an upstream node attempts to establish a new direct connection with a downstream node via the path folding mechanism as stated, the upstream node again decides to either send a new request or a termination message further upstream based on its updated demand after the recent connection establishment. This whole process continues until the downloader receives and processes the path folding message.

III. Behavior discrepancy exposed by path folding protocol

In this section, we present the key observation of protocol behavior discrepancy in the path folding protocol and our threat model.

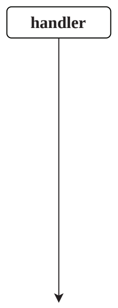


Fig. 2: Behavior of an intermediate node when handling a timeout event.

A. Key observation

Our investigation of the Freenet protocol reveals a key issue in the timeout handling for a path folding message. Specifically, intermediate nodes wait for a path folding request from their successor in the path and if they fail to receive the request within a specified time frame, they send a path folding termination message to their successor. However, a downloader the other hand, upon receiving a *AllReceived* message from the predecessor, the *handler* thread of the intermediate node waits for a notification from the *sender* thread for 2 minutes as well. There are three types of path folding messages, including the *FNPOpennetConnectDestinationNew* message to initiate a path folding request, the *FNPOpennetCompletedAck* message to terminate the path folding stage, and the *FNPOpennetCompletedTimeout* message to inform the predecessor of a timeout exception in the path folding.

We now explain how the protocol behavior discrepancy occurs between an intermediate and a downloader. As depicted in Figure 2, if the *sender* thread fails to receive any path folding message within 2 minutes, it notifies the *handler* thread to raise a timeout exception in the path folding stage. Then the *handler* thread informs the downloader by sending a *FNPOpennetCompletedTimeout* message and responds to its successor by sending a *FNPOpennetCompletedAck* message. If the *sender* thread continues to not receive any path folding messages after waiting for an additional 2 minutes, it tears down the connection with the successor as a fatal timeout exception occurred. However, the downloader only creates a *sender* thread that is incapable of emitting the *FNPOpennetCompletedAck* message to its successor if a path folding timeout exception occurs. Such distinct protocol behaviors can be exploited to break anonymity. Therefore, an attacker can deliberately cause a timeout in the path folding request and determine if the victim predecessor of the attacker's node is a downloader or not.

B. Threat model and assumptions

Our attack is to distinguish whether a victim node is a downloader. The attacker can deploy multiple Freenet nodes and manipulate messages transferred through the path so as to monitor and control the protocol behavior of the victim's predecessor and determine whether it is a downloader or not. This is a feasible assumption [9], as anyone can deploy any a timeout exception. If a second timeout exception occurs, the victim considers it as a serious connection issue and then tears down the connection with the attacker's node. To avoid detection, the malicious node proactively and promptly sends a path folding termination message to the predecessor during the second 2-minutes wait, thereby preserving the connection with the victim predecessor.

B. Imprecise timeout detection issue

A successful attack strongly relies on the *handler* thread of an intermediate node that is effectively notified by its *sender* thread so as to send out a path folding termination message expected by our malicious node. However, due to the complicated implementation of path folding timeout detection mechanism in Freenet, the timeout exception notification at an intermediate node may not be timely sent to the *handler* thread by the *sender* thread, and it causes the *handler* thread to fail to send the path folding *FNPOpennetCompletedAck* message to the successor. In this case, the attacker fails to receive the expected path folding termination message and it misleads she to recognize its predecessor as a downloader.

In practice, the timeout exception of waiting for a path folding message is detected by a *packet process* thread or a *timeout detection* thread. When a *sender* thread starts to wait for a path folding request, it constructs an event that includes the sender information of the expected path folding message, the expected message type, and the message waiting deadline, i.e., 2 minutes later, and inserts the event into an event queue. Once packets arrive at the UDP receiving buffer, the *packet process* thread is waken up to read packets and assemble messages. Then it inspects each event in the queue to determine if a message can match one of the events and also check whether the waiting time period of the event has finished. Since such timeout event detection triggered by the *packet process* thread strongly depends on the randomly arriving packets, it cannot precisely and promptly find out whether the waiting time period of an event exactly reaches 2 minutes. In addition, the *timeout detection* thread is a dedicated thread that is responsible for inspecting timeout exceptions for all of the *sender* threads at an intermediate node. It checks each event in the queue to determine if an event is timeout at each check time point. The time point of event timeout checking uses the timeout point of the upcoming timeout event in the queue. However, if the time interval between the current checking time and the timeout point of the upcoming timeout event is less than 1 second, the thread has to wait for 1 second to perform the timeout detection. In this case, the *timeout detection* thread cannot precisely detect the timeout event in 2 minutes either.

The *handler* thread uses timing method directly to precisely wait for the timeout event for 2 minutes. As a result, when the *sender* thread is waken up by the *packet process* thread or the *timeout detection* thread after 2 minutes so as to notify the *handler* thread, the *handler* thread is already waken up by itself and re-initiates a new path folding stage with its predecessor as it does not receive the notification from the *sender* thread in time. In this case, such an imprecise timeout

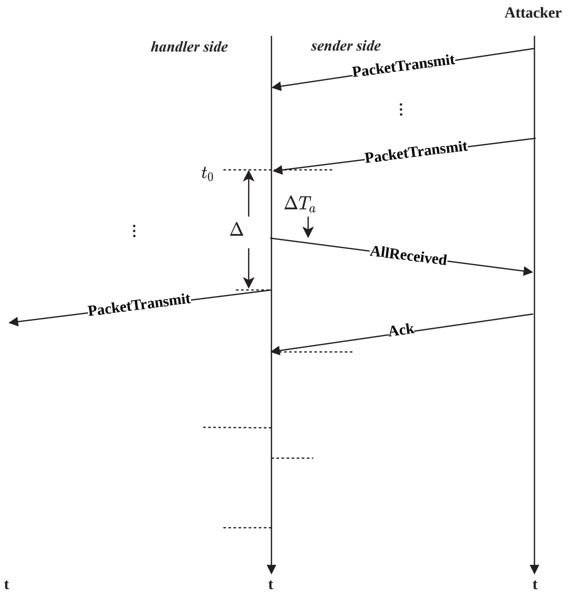


Fig. 4: Triggering condition for distinctive behaviour of an intermediate node.

detection issue happens and the *handler* thread ignores the delayed notification from the *sender* thread. Accordingly, the *handler* thread fails to send the path folding *FNPOpennetCompletedAck* message to the malicious successor, and causes the attacker to misidentify an intermediate node as a downloader.

To understand the challenge caused by the imprecise timeout detection issue, we use Figure 4 as an example to analyze the root cause of the attack failure. As we can see from this figure, the attacker controls a node that is a successor of an intermediate node who is the successor of a real downloader in the path. Let t_0 be the time at which the last *PacketTransmit* message of a CHK block is received by the intermediate node. Once receiving the last *PacketTransmit* message, the *sender* thread emits an *AllReceived* message to notify the attacker that the whole block is successfully received. Then the attacker sends back a packet to acknowledge the *AllReceived* message. Upon receiving the acknowledgment packet, the *sender* thread starts to wait for a path folding request from the malicious successor. We denote the time interval between the receiving time of the last *PacketTransmit* message of the CHK block and the sending time of the *AllReceived* message as ΔT_a. Let RTT_s be the time interval between the sending time of the *AllReceived* message and the receiving time of the acknowledgment packet. Denote the start waiting time and the end waiting time as t_s and t_1, respectively. Then, we can have

$$t_1 = t_0 + \Delta T_a + RTT_s + T + \Delta T. \tag{1}$$

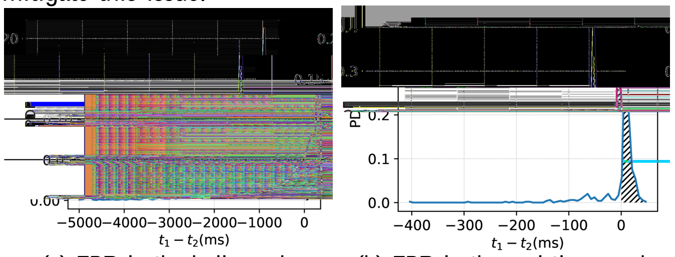
where T is the constant waiting time, i.e., 2 minutes, and ΔT is the error checking time interval caused by either of the two detection threads.

The *handler* thread at the intermediate node forwards the messages to the downloader. After the *handler* thread sends the last message of the CHK block, it can receive an *AllReceived* message as well and then check if the *sender* thread receives the acknowledgment packet from the attacker. If the acknowledgment packet is received, the *handler* thread starts to wait for a notification from the *sender* thread for 2 minutes. We denote the time interval between the receiving time of the last *PacketTransmit* message of the CHK block and the forwarding time of the last *PacketTransmit* message as ΔT_p. Let RTT_h be the time interval between the forwarding time of the last *PacketTransmit* message and the receiving time of the *AllReceived* message. Denote the start and end waiting time of the *handler* thread as t_h and t_2, respectively, where $t_h > t_s$. Finally, we can have

$$t_2 = t_0 + \Delta T_p + RTT_h + T. \tag{2}$$

Recall that the *handler* thread precisely sleeps the constant time T, i.e., 2 minutes, to wait for the notification from the *sender* thread.

According to our analysis, if the imprecise timeout detection issue happens, the end waiting time of the *sender* thread is later than the end waiting time of the *handler* thread, i.e., $t_1 > t_2$. It can cause a false positive downloader identification. Therefore, we define the *false positive rate* (FPR) of our downloader de-anonymization method as the probability that the event of $t_1 > t_2$ happens, i.e., $P(t_1 - t_2 > 0)$. We perform extensive empirical experiments to derive a probability density function (PDF) of $t_1 - t_2$ in two different downloading modes as shown in Figure 5a and Figure 5b. The shadow part in the figures represents the probability $P(t_1 - t_2 > 0)$. We further explain why the FPR in the bulk downloading mode is much smaller than that in the real-time downloading mode in Section V via experimental results. Next, we propose our solutions to mitigate this issue.



(a) FPR in the bulk mode (b) FPR in the real-time mode

Fig. 5: FPRs in two different downloading modes.

C. Optimization schemes

We analyze various factors in the false positive rate so as to improve the success rate. In fact, the attacker should make sure that the end waiting time of the *sender* thread is earlier than the end waiting time of the *handler* thread. Then the key factor of a success identification becomes

$$\begin{gathered} t_2 > t_1 \Rightarrow \\ t_0 + \Delta T_p + RTT_h + T > t_0 + \Delta T_a + RTT_s + T + \Delta T \Rightarrow \\ \Delta T_p - \Delta T_a + RTT_h - RTT_s - \Delta T > 0 \end{gathered} \tag{3}$$

According to this formula, we can attempt to increase ΔT_p and RTT_h and decrease ΔT_a, RTT_s, and ΔT in order to reduce the false positive identification. However, we can barely increase RTT_h that is determined by the network between the intermediate node and its predecessor. In addition, we cannot decrease the packet response time, i.e., ΔT_a, that is determined by the *scheduler* thread at the intermediate node. Therefore, we exploit the implementation features of Freenet protocol so as to actively manipulate the messages at the attacker's node that are sent to the victim node, to increase ΔT_p as well as decrease RTT_s and ΔT.

Scheme 1: Delaying the file block transmission of the intermediate node to increase ΔT_p. ΔT_p is the time interval between the receiving time of the last *PacketTransmit* message from the attacker at the *sender* thread side and the sending time of the last message to the predecessor at the *handler* thread side of the intermediate node. To increase ΔT_p, we exploit the packet sending and message processing mechanism in Freenet to send the *PacketTransmit* messages strategically at the attacker's node. Note that Freenet messages are enveloped into Freenet application layer packets for transferring. Each packet is limited to 1232 bytes to avoid IP fragmentation, and messages can be fragmented so as to fit into a packet. At the same time, Freenet node can only process a message after all fragments are received. Therefore, the intermediate node has to receive a complete *PacketTransmit* message and then forward it to the predecessor. If a message is not completely delivered by the attacker, the received part of the message is buffered in the memory at the intermediate node. Recall that Freenet employs 32 *PacketTransmit* messages to carry a 32 KB file block. As a result, we can deliberately segment each of the 32 *PacketTransmit* messages into two fragments, e.g., 1042 bytes and 10 bytes, where the 32 small fragments, i.e., 320 bytes, can be enveloped into one Freenet packet. After the Freenet packets that carry the 32 large message fragments are acknowledged by the intermediate node, we send the Freenet packet containing the 32 small message fragments. Once the intermediate node receives the packet, it can assemble the 32 whole messages in the queue and forward them to the predecessor packet by packet. In this way, we can significantly delay the delivery of the whole file block, thereby affecting the sending time of the last packet so as to increase ΔT_p.

Scheme 2: Acknowledging the *AllReceived* message promptly to decrease RTT_s. RTT_s consists of the transmission time of the *AllReceived* message and the acknowledgment packet over the network and the response time of the acknowledgment packet at the attacker's node. Let ΔT_{ack} be the response time of an acknowledgment. From the attacker's angle, we can only control the response time of the acknowledgment packet by acknowledging the *AllReceived* message as soon as the *AllReceived* message is received so as to minimize the ΔT_{ack}. For the sake of improving the node's throughput, an acknowledgment packet can wait 200ms at maximum before sending out so as to carry as much payload as possible. Consequently, in the worst case, the maximum delay of an acknowledgment packet at the attacker's node is 200 ms. It significantly increases RTT_s and raises the false positive rate. On the basis of this key insight, we can eliminate the response time period ΔT_{ack} by promptly responding the acknowledgment upon receiving the *AllReceived* message.

Scheme 3: Actively triggering timeout detection mechanism to decrease ΔT. Recall the *packet process* thread can also trigger the timeout detection when it receives a complete message. Once a message is received, the *packet process* thread inspects each event in the event queue to check if the event deadline has passed and whether the message matches the event. Accordingly, we can actively send forged messages to the intermediate node to force the timeout detection via *packet process* thread, even the messages can not match any event. In practice, we send a *FNPOpennetCompletedAck* message with a random UID. Ideally, our message shall arrive at the intermediate node after but very close to the event deadline. So the timeout is discovered by the *packet process* thread and the *sender* thread can be timely waken up to notify the *handler* thread. For the attacker, the estimated sending moment thus can be T plus a small delay time. We try to estimate the moment to send our delayed forged message to the intermediate node so as to efficiently squeeze ΔT with effort. A number of empirical experiments are performed in Section V to select the delay time for sending the forged message.

V. Evaluation

In this section, we conduct real-world experiments to demonstrate the feasibility and effectiveness of our de-anonymization attack against downloaders in Freenet. To uphold legality and ethics, we restrict our experiments to only file block retrievals initiated by our own deployed Freenet nodes.

A. Experimental setup

We perform two groups of experiments respectively to evaluate the false positive rate (FPR) and true positive rate (TPR) of our attack. To estimate the FPR, we deploy three nodes including a downloader, an intermediate node and a malicious node in the real-world Freenet network. Each deployed node follows the rule of Freenet to connect with other nodes automatically, excepting for the connection we added manually among them. We use ΔRTT to represent $((RTT_h - \Delta T_a^0) - (RTT_s - \Delta T_{ack}))$, which describes the round-trip time difference of two network intervals (downloader $ intermediate, and intermediate $ malicious). Both the downloader and the malicious node can be respectively hosted on a virtual private server (VPS) in candidate locations including Paris, France, Tokyo, Japan, and Chicago, USA with 2 virtual cores and 2G memory, while the intermediate node is hosted on a VPS in New York, USA with 4 virtual cores and 8G memory. We install Ubuntu 20.04 and OpenJDK 11 JRE on all VPSs. The source code of Freenet 0.7.5 (Build 1494) is revised and recompiled to incorporate our attack. To estimate the TPR, we set up a downloader and a malicious node on separate VPSs. The downloader and the malicious node are hosted on VPSs in Paris, France and New York, USA, respectively. Other setup is identical to the experiments setup in determining the FPR.

We implement our basic de-anonymization approach and three enhanced schemes by revising the Freenet source code of the malicious node. To enable file block downloading, we generate a customized CHK block and add it into our malicious node. Freenet node integrates a telnet-based local control service which handles control commands such as uploading or downloading a block. Consequently, we connect to the local control service of our malicious node using a telnet terminal and then send a CHK file block content, e.g., a string sequence, to the service to require the generation of a 32 KB CHK block. To minimize the impact on Freenet, the source code of the malicious node is altered so that the malicious node only stores the block locally without network propagation.

In the FPR evaluation experiments, we revise the Freenet source code of the downloader to make it download the file block via our intermediate node. To this end, we first set the downloader's Freenet configuration to add our intermediate node as its peer and modify the source code to ensure that this peer node is not replaced. We perform the same configuration on the intermediate node, making the downloader its peer. Then we revise the source code of the downloader to use the intermediate node as its successor in the path. Through downloader's control service, we issue download requests to fetch CHK blocks generated by the malicious node. The necessary information regarding the block retrieval requests is recorded in each node to aid in analysis.

In the TPR evaluation experiments, we revise the source code and configure both the downloader and malicious nodes to add each other as a peer. Then the downloader directly downloads randomly generated file blocks from the malicious node, so that we can evaluate whether our malicious node can identify the downloader using our attack.

B. Experimental results

The effectiveness of the de-anonymization attack is measured by the TPR and FPR. Our original attack is referred as the *baseline attack*, and the attack with all three enhanced schemes is referred to as the *enhanced attack*. The default value of output bandwidth of a node is 320KB/s, The default ΔRTT is 0 by hosting both the malicious node and the downloader in Paris, France and the intermediate node in New York, USA. We estimate the TPR and FPR by changing the value of one of these variables while keeping others at default.

We test the attack with five different output bandwidth limits of the downloader and two download modes, bulk mode and real-time mode. To calculate the TPR, the downloader directly connects to the malicious node and downloads 500 randomly generated CHK blocks for each of the 10 (5 bandwidth limits$\times 2$ modes) settings, namely a total of 5,000 downloads. The TPRs of both the baseline and enhanced attacks in both modes are 100%, which show the distinct path folding protocol behavior can effectively help identify the downloader.

TABLE I: FPR (%) of identifying a downloader.

Output bandwidth limit (B/s)	Bulk mode		Real-time mode	
	Baseline	Enhanced	Baseline	Enhanced
16K	1.80	0	1.60	0
32K	5.00	0	27.60	0
320K	12.20	0	81.20	1.80
1M	12.40	0	81.80	2.20
2M	10.20	0	79.80	1.80

Table I illustrates the FPR evaluations for both the baseline attack and the enhanced de-anonymization attack using five different output bandwidth limits and two downloading modes.

The evaluation is conducted by applying 5,000 downloads (same composition as in the TPR experiments) for either of the two attacks. As shown, the FPRs of both attacks in the bulk mode are significantly much lower compared to those in the real-time mode. In fact, the default maximum waiting time of the *PacketTransmit* messages in the queue at the intermediate node are 5s for the bulk mode and 100ms for the real-time mode. The longer waiting time for the *PacketTransmit* messages in the bulk mode increases ΔT_p, resulting in a lower FPR. Moreover, the enhanced attack successfully reduces the FPR by increasing ΔT_p and decreasing both RTT_s and ΔT. The FPRs approach zero in the bulk mode with all five output bandwidth limits and are also nearly zero in the real-time mode with output bandwidth limits of 16KB/s and 32KB/s. These demonstrate that our theoretical analysis in uncovering the underlying cause of the false positive detection is effective and the three proposed enhancement techniques can be successfully implemented in practice.

In order to find an appropriate delay time for sending the forged message in Scheme 3, extensive empirical experiments are conducted. Figure 6 illustrates the relationship between the FPR and delay time. The results show that the FPR reaches 6% when the delay time of sending the message is 1ms in the bulk mode, while in the real-time mode, the FPR is around 65% when the delay time is set to 3ms. Accordlly, to achieve a lower FPR, we use a delay time of 1ms and 3ms in Scheme 3 for the bulk and real-time modes, respectively.

To assess the individual contribution of the three different schemes, we perform four sets of experiments using the baseline attack and three enhanced attack variants that respectively employ one of the three schemes. In each experiment set, we configure the output bandwidth of the intermediate node to 320KB/s and then control the downloader to fetch 500 randomly generated block files from our attacker's node for each of the 8 (4 attacks×2 modes) experimental settings, thus totally 4,000 downloads. Figure 7 illustrates the comparison of the FPRs among these attacks in two different download modes. As seen in the figure, all these schemes effectively reduce the FPR. The performance of the scheme 1 in the bulk mode outperforms that of the other two, while the performance of scheme 2 achieves the best results in the real-time mode.

To better understand the underlying reasons of the results in Figure 7, we further analyze the PDF and the average time of ΔT_p, ΔT_{ack}, and ΔT_p in the different schemes. Figure 10a illustrates the PDF of ΔT_p in the bulk mode for both the baseline attack and scheme 1. We observe that average of ΔT_p considerably grows when $\Delta T_p \le 1500$ms with scheme 1 applied. Note that the cases when $\Delta T_p > 1500$ms is less important since the false positives are almost impossible to happen when ΔT_p is large. Then we compute the average value of ΔT_p within 1500ms and find that ΔT_p of scheme 1 in the bulk mode is around 250ms larger than that of the baseline attack at average as shown in Table II. In comparison, scheme 2 and scheme 3 only decrease ΔT_{ack} and ΔT by around 160ms and 4ms, respectively. Figure 11a and Figure 12a show the PDF of ΔT_{ack} and ΔT in the bulk mode. Although these two schemes also significantly decrease the ΔT_{ack} and ΔT, the total squeezed time is still smaller than the time decreased by using scheme 1. Therefore, according to the analysis in Equation (3), the scheme 1 is the crucial contributor that increases the probability of $t_2 - t_1$ so as to substantially reduce the FPR to a value close to zero. That is the reason why it performs the best in the bulk mode.

TABLE II: Average time (ms) of ΔT_p, ΔT_{ack}, and ΔT_p.

	Bulk mode			Real-time mode		
	$\Delta T_p \uparrow$	$\Delta T_{ack} \downarrow$	$\Delta T \downarrow$	$\Delta T_p \uparrow$	$\Delta T_{ack} \downarrow$	$\Delta T \downarrow$
Baseline	573.87 (within 1500)	161.07	7.19	78.98	161.39	8.91
Scheme 1	822.38 (within 1500)			115.98		
Scheme 2		0.33			0.24	
Scheme 3			4.33			2.83

$\uparrow$ indicates the larger the better, $\downarrow$ indicates the smaller the better.

We then demonstrate why scheme 2 in the real-time mode

in

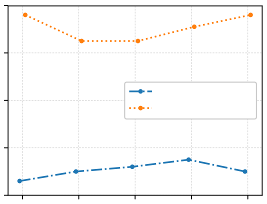


Fig. 6: FPR versus delay time of the forged message.

Fig. 7: FPR comparisons using different schemes.

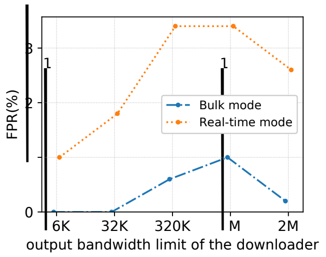


Fig. 8: FPR under different bandwidth limits of the victim.

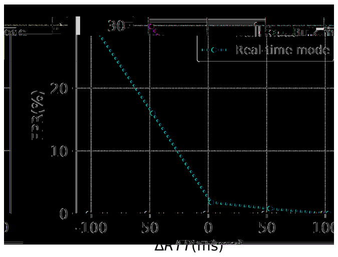
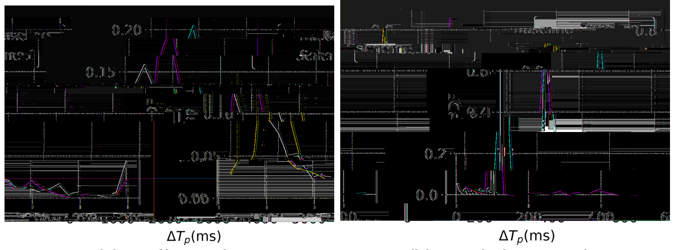
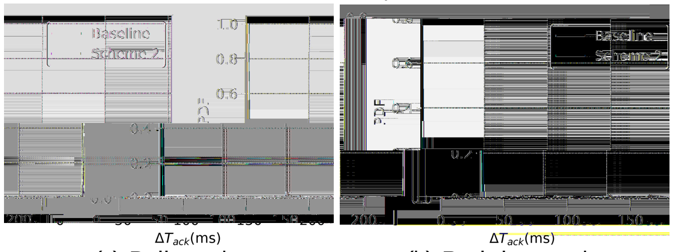


Fig. 9: FPR under different ΔRTT.



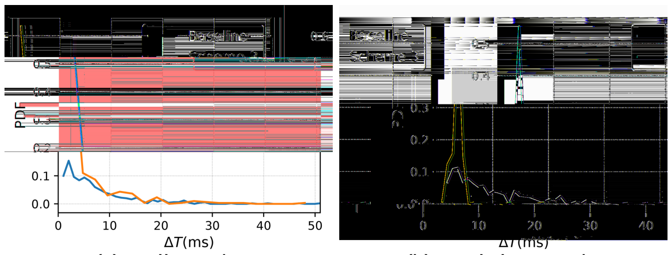
(a) Bulk mode. (b) Real-time mode.

Fig. 10: Increase of ΔT_p using scheme 1.



(a) Bulk mode. (b) Real-time mode.

Fig. 11: Decrease of ΔT_{ack} using scheme 2.



(a) Bulk mode. (b) Real-time mode.

Fig. 12: Decrease of ΔT using scheme 3.

[14], etc.. Meanwhile, various de-anonymization attacks are investigated to uncover vulnerabilities in these systems. These attacks can be classified into: anonymous originator traceback attacks [15], [16], [17], [18], [19], [7], [20], [21], [22] and identification attacks [23], [24], [25], [8], [9].

In Freenet, Baumeister *et al.* [26] investigate a routing table insertion (RTI) attack by exploiting the peer replacement mechanism. Tian *et al.* [6], [7] propose a traceback attack

References

[1] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proceedings of the 2000 International Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, Berlin, 2001. Springer-Verlarg.

[2] Ian Clarke, Scott.G. Miller, Theodore W. Hong, Oskar Sandberg, and Brandon Wiley. Protecting free expression online with freenet. *IEEE Internet Computing*, 6(1):40–49, January 2002.

[3] Ian Clarke, Oskar Sandberg, Matthew Tosel, and Vilhelm Verendel. Private communication through a network of trusted connections: The dark freenet. https://hyphanet.org/assets/papers/freenet-0.7.5-paper.pdf, 2010.

[4] Hao Zhang, Yonggang Wen, Haiyong Xie, and Nenghai Yu. *Distributed Hash Table: Theroy, Platforms and Applications*. Springer, 2013.

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

[6] Guanyu Tian, Zhenhai Duan, Todd Baumeister, and Yingfei Dong. A traceback attack on freenet. In *Proceedings of the 2013 IEEE International Conference on Computer Communications*, pages 1797–1805, 2013.

[7] Guanyu Tian, Zhenhai Duan, Todd Baumeister, and Yingfei Dong. A traceback attack on freenet. *IEEE Transaction on Dependable and Secure Computing*, 14(3):294–307, 2017.

[8] Brian N Levine, Marc Liberatore, Brian Lynn, and Matthew Wright. Statistical detection of downloaders in freenet. In *Proceedings of the IEEE International Workshop on Privacy Engineering*, pages 25–32, may 2017.

[9] Brian N. Levine, Marc Liberatore, Brian Lynn, and Matthew Wright. A forensically sound method of identifying downloaders and uploaders in freenet. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1497–1512, New York, 2020. ACM.

[10] Oskar Sandberg and Ian Clarke. The evolution of navigable small-world networks. *ArXiv*, abs/cs/0607025, 2006.

[11] Arne Babenhauserheide. Freenet build 1497:fix severe path folding vulnerability. https://www.hyphanet.org/freenet-build-1497-fix-severe-path-folding-vulnerability.html, 2023.

[12] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, Berkeley, 2004. USENIX.

[13] Lars Schimmer. Peer profiling and selection in the i2p anonymous network. In *Extended Abstracts of the Fourth Privacy Enhancing Technologies Convention*, Technische Berichte, pages 59–70, Germany, 2009. Technische Universität Dresden.

[14] Tomas Isdal, Michael Piatek, Arvind Krishnamurthy, and Thomas Anderson. Privacy-preserving p2p data sharing with oneswarm. *ACM SIGCOMM Computer Communication Review*, 40(4):111–122, August 2010.

[15] Kevin Bauer, Damon McCoy, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Low-resource routing attacks against tor. In *Proceedings of the 2007 ACM workshop on Privacy in electronic society*, pages 11–20, 2007.

[16] Timothy G Abbott, Katherine J Lai, Michael R Lieberman, and Eric C Price. Browser-based attacks on tor. In *Proceedings of the International Workshop on Privacy Enhancing Technologies*, pages 184–199. Springer, 2007.

[17] Xinwen Fu, Zhen Ling, J Luo, W Yu, W Jia, and W Zhao. One cell is enough to break tor's anonymity. In *Proceedings of the Black Hat Technical Security Conference*, pages 578–589, 2009.

[18] Zhen Ling, Junzhou Luo, Wei Yu, Xinwen Fu, Dong Xuan, and Weijia Jia. A new cell counter based attack against tor. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, pages 578–589, New York, 2009. ACM.

[19] Zhen Ling, Junzhou Luo, Wei Yu, Xinwen Fu, Weijia Jia, and Wei Zhao. Protocol-level attacks against tor. *Computer Networks*, 57(4):869–886, March 2013. https://doi.org/10.1016/j.comnet.2012.11.005.

[20] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. DeepCorr: Strong flow correlation attacks on tor using deep learning. In *Proceedings of the 25th ACM Conference on Computer and Communications Security (CCS '18)*, November 2018.

[21] Florentin Rochet and Olivier Pereira. Dropping on the edge: Flexibility and traffic confirmation in onion routing protocols. *Proceedings on Privacy Enhancing Technologies*, 2018(2):27–46, 2018.

[22] Zhen Ling, Junzhou Luo, Danni Xu, Ming Yang, and Xinwen Fu. Novel and practical sdn-based traceback technique for malicious traffic over anonymous networks. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 1180–1188, 2019.

[23] Matthew K. Wright, Micah Adler, Brian Neil Levine, and Clay Shields. The predecessor attack: An analysis of a threat to anonymous communications systems. *ACM Transaction on Information and System Security*, 7(4):489–522, November 2004.

[24] Swagatika Prusty, Brian Neil Levine, and Marc Liberatore. Forensic investigation of the oneswarm anonymous filesharing system. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, page 201–214, New York, NY, USA, 2011. Association for Computing Machinery.

[25] Albert Kwon, Mashael AlSabah, David Lazar, Marc Dacier, and Srinivas Devadas. Circuit fingerprinting attacks: Passive deanonymization of tor hidden servicheE-4291 (Comc2idden)-279.9-hidd54 tor