Time will Tell: Large-scale De-anonymization of Hidden I2P Services via Live Behavior Alignment

Hongze Wang[†], Zhen Ling[†*], Xiangyu Xu[†], Yumingzhi Pan[†], Guangchi Liu[†], Junzhou Luo[†‡] and Xinwen Fu[§]
[†]Southeast University, Email: {wanghongze, zhenling, xy-xu, pymz, gc-liu, jluo}@seu.edu.cn
[‡]Fuyao University of Science and Technology
[§]University of Massachusetts Lowell, Email: xinwen_fu@uml.edu

***Abstract*—I2P (Invisible Internet Project) is a popular anonymous communication network. While existing de-anonymization methods for I2P focus on identifying potential traffic patterns of target hidden services among extensive network traffic, they often fail to scale effectively across the large and diverse I2P network, which consists of numerous routers. In this paper, we introduce I2PERCEPTION a low-cost approach revealing the IP addresses of I2P hidden services. In I2PERCEPTION, attackers deploy floodfill routers to passively monitor I2P routers and collect their *RouterInfo*. We analyze the router information publication mechanism to accurately identify routers' join (i.e. on) and leave (i.e. off) behaviors, enabling fine-grained live behavior inference across the I2P network. Active probing is used to obtain the live behavior (i.e., on-off patterns) of a target hidden service hosted on one of the I2P routers. By correlating the live behaviors of the target hidden service and I2P routers over time, we narrow down the set of routers matching the hidden service's behavior, revealing the hidden service's true network identity for de-anonymization. Through the deployment of only 15 floodfill routers over the course of eight months, we validate the precision and effectiveness of our approach with extensive real-world experiments. Our results show that I2PERCEPTION successfully de-anonymizes all controlled hidden services.** [1]

I. Introduction

I2P (Invisible Internet Project) is a widely used anonymous communication network, supporting activities such as website browsing, email, file sharing, Internet Relay Chat (IRC), and cryptocurrency access, with around 45,000 active routers and over 15,000 daily users and services [2]. As a decentralized peer-to-peer system, I2P enables anonymous data transmission through routers that voluntarily share bandwidth and storage. A subset of routers, referred to as *floodfill* routers, maintain a distributed database containing access information for all routers and hidden services on I2P. Like Tor [3], I2P uses multilayer encryption and multihop tunnels to secure communication, ensuring that no single router can correlate client-service interactions or identify the hosting router of a specific hidden service, for the sake of privacy and secure communication.

To assess the security and limitations of anonymous networks, a large volume of prior work has investigated deanonymizing methods. These efforts have primarily concentrated on the Tor network, one of the most prevalent anonymous communication systems, and are mainly based on the traffic correlation approaches [4]–[35]. Such approaches typically require the attacker to first use techniques like DoS attacks to occupy key positions in the target hidden service's circuit [36]–[38], in order to conduct traffic correlation. However, while I2P employs a multi-hop tunnel architecture comparable to Tor's circuit for achieving anonymity, it adopts a different approach from Tor for selecting participants for a hidden service's tunnel. Given the current scale of the I2P network, it is exceedingly difficult for an attacker to successfully control the routers participating in a target hidden service's tunnels, thereby making traffic correlation unachievable.

Although I2P's network communication mechanisms and scale seemingly allow a hidden service to hide well within the 'crowd', we demonstrate that the online and offline patterns of a hidden service, referred to as its live behavior, can still serve as a powerful side-channel for compromising its anonymity. Our approach relies on two key observations. First, an I2P hidden service is hosted on an I2P router, resulting in its live behavior to closely align with that of its hosting router. Second, user behavior in peer-to-peer (P2P) systems, including I2P, is inherently diverse [39]–[43], causing significant variations in router behavior due to the heterogeneity of user activity. Therefore, by identifying the router whose live behavior most closely aligns with that of a target hidden service, we can accurately infer the location of the hosting router and de-anonymize the hidden service.

However, the practical implementation of the aforementioned deanonymization method presents significant challenges. First, achieving large-scale, stealthy, and fine-grained monitoring of I2P routers' live behavior simultaneously is highly difficult. Active probing of approximately 45,000 routers daily is impractical due to excessive overhead, the risk of exposing probing intentions, and potential degradation of network performance. Passive monitoring faces grand challenges too, as only *RouterInfo* (which contains the router's contact information and is periodically published to floodfill routers) is passively obtainable, and I2P employs obfuscation mechanisms to prevent *RouterInfo* from revealing detailed behavior information. Second, even if the router's live behavior could be inferred from *RouterInfo*, the large scale of I2P makes it exceedingly difficult to collect complete data in a cost-effective manner. Accurately inferring a router's live behavior under incomplete data conditions poses a significant challenge.

In this paper, we introduce I2PERCEPTION, a large-scale

[*] Corresponding author: Prof. Zhen Ling of Southeast University, China.

[1] Please refer to the extended version [1] of the paper for more technical details.

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.240114
www.ndss-symposium.org

deanonymization approach designed to reveal the real IP address of a target I2P hidden service. First, we leverage only 15 floodfill routers to passively collect *RouterInfo* at low cost from all I2P routers to create a *RouterInfo* trace for each router. Using these traces, we exploit the periodic *RouterInfo* publication patterns across different I2P implementations to achieve coarse-grained online session inference of each router. We then investigate the *RouterInfo* publication mechanisms during the join and leave periods of routers to precisely identify their join and leave behaviors, achieving fine-grained router behavior inference. To mitigate the impact of incomplete data due to our low-cost data collection strategy, we propose a data-recovery-based online session complement method. Next, we apply active probing to infer the live behavior of the target hidden service. Finally, we introduce a live behavior correlation method to measure the similarity between the inferred live behaviors of routers and the target hidden service, allowing us to identify the host router of the hidden service.

Our major contributions can be summarized as follows:

- We propose a large-scale and long-lasting I2P hidden server deanonymization framework, I2PERCEPTION, based on the similarity of the live behavior of the target hidden service and I2P routers, which can be used to deanonymize any target hidden service in the I2P network within a proper monitoring window.
- We design methods that can accurately identify the join and leave behaviors of routers by delving into the *RouterInfo* publication mechanism during the startup and offline periods. We introduce an online session recovery method to mitigate the impact of incomplete data caused by the low-cost data collection strategy, achieving highly accurate inferred live behaviors for all routers in the I2P network.
- To demonstrate the feasibility and effectiveness of I2PERCEPTION, we conduct extensive real-world experiments by deploying 15 floodfill routers and 10 I2P routers hosting hidden services over a period of approximately eight months. The accuracy of deanonymizing our controlled hidden services approaches nearly 100%.

The key novelty of this work is that we exploit a user's characteristic on–off behavior when operating an I2P hidden service and introduce new measurement techniques that accurately capture that pattern to deanonymize the service. The scientifically generalizable takeaway is that user behavior can serve as a measurable side channel that leaks identity.

Responsible Disclosure. Upon discovering the vulnerabilities, we disclosed our findings to the I2P maintainers to support timely mitigation. Our work was acknowledged by the I2P project, and as of this submission, the reported issues have been addressed [44].

II. Background

In this section, we first provide an overview of the I2P network structure, and then present its communication mechanism.

A. Architecture of I2P

The Invisible Internet Project (I2P) is a decentralized peer-to-peer communication system enabling anonymous data transmission. As shown in Fig. 1, the I2P network comprises four key components: I2P routers, I2P clients, hidden services, and netDB [45].

I2P router. I2P routers form the backbone of I2P's communication infrastructure. Upon installation, router functionality is enabled by default. During its initial startup, the router contacts specific bootstrap sites to discover other routers, thereby joining the I2P network and relaying data for other participants. Routers can operate in either *floodfill* mode or *non-floodfill* mode. While non-floodfill routers primarily relay data, floodfill routers also gather and maintain information about all routers in the I2P network, including both *floodfill* and *non-floodfill* routers.

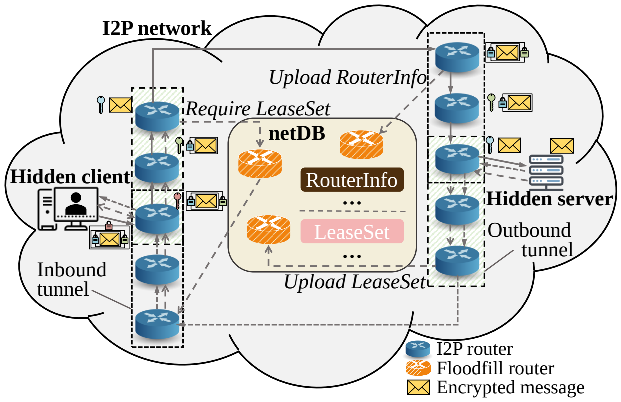


Fig. 1: I2P Architecture.

I2P client. The I2P software includes several default application clients, such as an IRC client and a BitTorrent client. The software also supports an HTTP proxy, allowing various clients to connect to the I2P network through the router. A router that hosts a client, referred to as a *host router*, relays client data via multiple two-hop, unidirectional inbound and outbound tunnels.

I2P hidden service. I2P provides a web service for users to deploy hidden web services on the network. Additionally, an I2P router supports a simple anonymous messaging (SAM) protocol, enabling various services to interact with the host router. Like an I2P client, a hidden service uses the host router to establish inbound and outbound tunnels to relay its data.

netDB. The netDB is a distributed database composed of all floodfill routers, maintaining information about I2P routers and hidden services. Floodfill routers are organized using a distributed hash table (DHT) based on the Kademlia algorithm [46]. The netDB stores two types of data: *RouterInfo* and *LeaseSet*. *RouterInfo* contains router details, such as IP address, port, and public key. An I2P router periodically selects a floodfill router to upload its *RouterInfo*, with the floodfill router returning an acknowledgment message confirming the upload. A hidden service or client records its details, including a public key and the entry router of its inbound tunnel in the *LeaseSet* data. Hidden services upload their *LeaseSet* to the netDB, while clients send their *LeaseSet* to a target hidden service via the outbound tunnel when interacting with the service. A

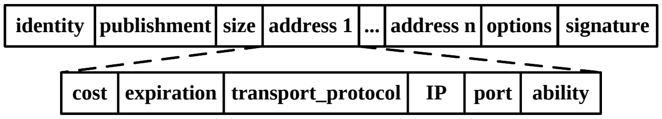


Fig. 2: Format of RouterInfo data.

LeaseSet expires based on the service's configuration, while *RouterInfo*, expires after one hour unless updated.

B. Garlic Routing

I2P uses garlic routing to facilitate anonymous communication between hidden services and clients. Routers obtain *RouterInfos* from the netDB (floodfill routers) to discover other routers. A hidden service or client relies on its host router to select suitable routers and establish paired, unidirectional, two-hop inbound and outbound tunnels for data transmission, as shown in Fig. 1. Each tunnel consists of three I2P routers, including the host router, and the data transmitted through these tunnels are called garlic messages. To connect to a hidden service, a client derives the service's I2P address and queries the netDB to retrieve its *LeaseSet*, which contains the entry router for the service's inbound tunnel. The client then selects an outbound tunnel to establish the connection, sending its *LeaseSet* to the hidden service. By learning the entry router for the client's inbound tunnel from the *LeaseSet*, the service initiates a connection to the client's inbound tunnel through its outbound tunnel. Bidirectional communication is achieved through the concatenated four-hop tunnels. Each router in these tunnels can only identify the IP addresses of its immediate predecessor and successor, making it difficult to correlate client-server communication.

C. RouterInfo

Structure. Fig. 2 illustrates the structure of the *RouterInfo*. The first field, *Identity*, uniquely identifies the router within the I2P network and is generated upon the router's initial launch. The *publishment* field records the creation time of the *RouterInfo*, while the *size* field indicates the number of associated *RouterAddress* entries. Each *RouterAddress* contains essential connection details, such as *cost*, *expiration* (deprecated), I2P transport protocol (NTCP or SSU), IP address, port, and an *ability* field. The *cost* value is assigned by the router, where a lower cost increases the likelihood of being chosen for connections. The *ability* field indicates whether the router offers a *peer testing* service (with a 'B' flag) or an *introducing* service (with a 'C' flag). The 'B' flag means the router can test reachability, while the 'C' flag indicates it can function as a reverse proxy for NAT-bound routers. The final two fields, *signature* and *options*, ensure integrity and store the router's profile, including operational mode, bandwidth, congestion level, port reachability, and version number. Floodfill routers are marked with an 'f' flag in the *options* field, distinguishing them from non-floodfill routers.

Propagation. I2P implements a specific mechanism for uploading and propagating *RouterInfos*. Each router selects a target floodfill router to upload its *RouterInfo*, based on a parameter known as the *Routingkey*, which is computed using the *SHA256* hash algorithm as follows:

$$Routingkey = Hash(Hash(identifier) + date) \qquad (1)$$

Here, *date* denotes the formatted timestamp of the *RouterInfo* publication, and *identifier* refers to the router's unique ID within the I2P network. The router then selects the floodfill router whose identifier has the smallest *XOR* distance to its own *Routingkey* [46]. However, since the router lacks global knowledge of all floodfill routers in the network, there may exist others with identifiers even closer to its *Routingkey*. To address this, each floodfill router is also responsible for further propagating *RouterInfos*. After extracting the originating router's identifier and publication time, a floodfill router recomputes the corresponding *Routingkey* and forwards the *RouterInfo* to three additional floodfill routers from its known set, based on the same selection rule. Forwarding is skipped if no suitable floodfill routers are available or if the *RouterInfo* has already been processed by the router. This propagation mechanism ensures that each *RouterInfo* data could be received by numerous floodfill routers upon publication.

III. Deanonymizing I2P via Live Behavior

A. Problem Statement

Our objective is to stealthily uncover the real IP addresses of target I2P hidden services. Specifically, we aim to identify the I2P router hosting the hidden service. However, this task can be challenging. As depicted in §II-B, the multi-hop *garlic routing* ensures that both the client and the server only know the IP addresses of their respective tunnel entry routers, without revealing each other's actual IP addresses. Additionally, each participant router within these tunnels is only aware of the IP addresses of its predecessor and successor. This design prevents routers from not only correlating the communication relationship between the client and the hidden services but also identifying whether a specific hidden service is hosted on an I2P router.

B. Threat Model

We make three key assumptions for our method. First, an attacker can deploy several floodfill routers to join the I2P network and collect *RouterInfo* data from other routers. This is feasible due to I2P's open and decentralized nature, which allows anyone to deploy routers and participate. By modifying the configuration, the attacker can enable floodfill mode on these routers to monitor routers' live behavior using their uploaded *RouterInfos*. Second, the attacker targets a specific hidden service for deanonymization, which is accessible without restrictions. The target server cannot distinguish between the attacker's access and that of other users, preventing it from revealing the attacker's IP address. This allows the attacker to observe the service's live behavior. Third, the target hidden service is not hosted on a router in *hidden mode*. Collecting hidden routers' *RouterInfo* is extremely challenging, making them nearly impossible to monitor. However, this assumption is reasonable since routers operate in *non-hidden* mode by default and, according to our long-term measurements of the I2P network, we believe that hidden routers constitute a very small fraction of the network. The difficulty in obtaining their *RouterInfo* arises for two reasons: (1) floodfill routers do not broadcast the *RouterInfo* of hidden routers; and (2) hidden routers reject all inbound connections and can only be detected by routers, including floodfill routers, to which they initiate outbound connections. Over the monitor of one year, we deployed 15 floodfill routers, added 5 more for two months, and operated an additional 10 non-floodfill routers for eight months. Throughout this period, none of our controlled routers were contacted by any hidden routers.

C. Basic Idea

We leverage two fundamental observations that arise from the characteristics of I2P's design to de-anonymize a target hidden service. First, since each hidden service is hosted on an I2P router, its live behavior mirrors that of the hosting router. This means that by identifying the router's live behavior, we can infer the hidden service's behavior. In addition, we observe that I2P routers periodically upload their *RouterInfo* to the netDB, a regular pattern that can potentially be used to infer their live behaviors.

Second, the live behaviors of I2P routers exhibit significant variation. Previous studies [47] show that only 31.15% of I2P routers remain active for more than 30 days, with many exhibiting intermittent availability. Other research [39], [40], [42] on P2P networks indicates that user behavior is inherently diverse, with substantial behavioral differences among users. This diversity in router behaviors further reduces the anonymity set — the group of I2P routers with similar behaviors — making it easier to distinguish individual routers.

By combining these observations, an attacker can monitor the live behaviors of I2P routers and correlate them with the live behavior of the target hidden service. Due to behavioral divergence among routers, the overlap between their live behaviors naturally decreases over time. As the monitoring period extends, the number of routers whose behavior matches the target service gradually decreases. This allows the attacker to progressively narrow down the set of potential hosting routers, ultimately achieving deanonymization.

D. Challenges and Solutions

While deploying the de-anonymization approach is theoretically straightforward, several practical challenges arise during its implementation. We address these challenges as follows.

(C-I) Large-scale Analysis of Routers' Fine-grained Live Behaviors. A typical method for inferring the fine-grained live behavior of an I2P router involves periodically and actively probing its availability. An attacker can leverage the router's IP address and port information from the *RouterInfo* to perform port scanning. However, this approach introduces significant overhead, especially when monitoring a large number of I2P routers simultaneously (approximately 45,000 active routers each day [2]). Furthermore, active probing risks exposing the attacker's intentions, potentially alerting the targets and prompting evasive actions. Consequently, passive monitoring approaches, which can unobtrusively monitor the live behavior of a large number of routers, are crucial for large-scale analysis.

However, passive monitoring poses specific challenges due to limited data availability for inferring router behavior. While I2P routers periodically upload *RouterInfo* to the netDB—an activity that can be leveraged to infer coarse-grained live behavior—the system intentionally introduces randomness into the intervals between successive routine uploads to obfuscate predictability. This randomness undermines the reliability of coarse-grained inference methods, increasing their susceptibility to errors, particularly when routers experience short-term offline periods. Consequently, fine-grained identification of online sessions, which is capable of precisely determining router online and offline times, remains a nontrivial task.

(S-I) RouterInfo-publication-based Live Behavior Inference (§IV-C). Through a detailed investigation of the *RouterInfo* publication mechanism, we propose a passive, low-cost, large-scale monitoring method for identifying fine-grained live behavior. Regardless of the I2P implementation (Java or C++), an online I2P router is designed to perform multiple tasks to generate and upload its *RouterInfo* to the netDB (i.e., floodfill routers), thereby informing other routers in the network of its current status, such as reachability and load. We focus on the periodically executed tasks and analyze the characteristics of the generated *RouterInfo* to identify which publications are associated with these tasks. By only passively deploying 15 floodfill routers, we can infer the implementation of the I2P router based on the periodic update intervals and further identify gaps between routine publications that are longer than expected, and interpret them as offline periods. To achieve fine-grained identification of temporary offline behavior, we find that I2P routers display distinctive *RouterInfo* publishing patterns when going online and offline, allowing us to determine the exact start and end times of each online session.

(C-II) Accuracy of Low-cost Live Behavior Identification. Our live behavior inference method for I2P routers relies heavily on *RouterInfos* collected by only 15 controlled floodfill routers. However, gathering complete *RouterInfos* from a large number of routers may require deploying hundreds or even thousands of additional floodfill routers, as approximately 4,000 already operate daily, servicing around 45,000 active routers [2]. While deploying sufficient floodfill routers can capture *RouterInfos* to accurately infer online sessions, this approach introduces substantial overhead and may compromise stealth. To reduce costs, the attacker may limit the number of floodfill routers, resulting in incomplete *RouterInfo* collection, which complicates accurate online session identification.

Additionally, while the *RouterInfo* propagation mechanism (Appendix A) enables the attacker to collect most network-wide data with a limited number of floodfill routers, network churn––the continuous arrival and departure of thousands of routers—— can cause *RouterInfos* to be uploaded to different floodfill routers or propagated via different paths over time. This variability leads to partial *RouterInfo* collection for each I2P router, reducing the accuracy of live behavior inference and causing mismatches when comparing router behaviors to that of the tar-

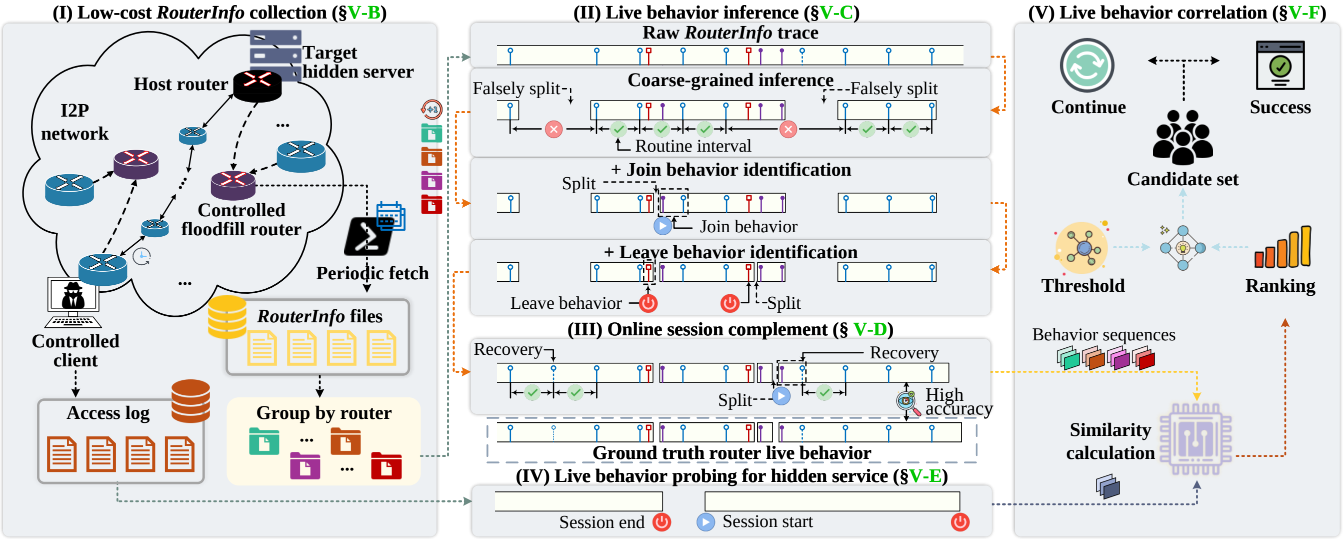


Fig. 3: I2PERCEPTION Overview.

get hidden service. Mitigating the impact of incomplete *RouterInfo* data on inference accuracy remains a significant challenge.

(S-II) Data-recovery-based Session Complement (§IV-D). We propose a router behavior complement method to mitigate the impact of incomplete *RouterInfo* data. Each online session is expected to include explicit start-up and offline behaviors. For incomplete sessions, we first determine whether the issue arises from a single session being erroneously split into two due to missing *RouterInfo*, or from absent startup or shutdown indicators. Based on this determination, we adopt a context-aware strategy to insert the missing *RouterInfos* at suitable points, allowing for accurate reconstruction of the session.

IV. Design of I2Perception

In this section, we first present the workflow of I2PERCEPTION and then introduce its five phases in detail. A theoretical analysis is provided at the end to show why the target hidden service can be uniquely identified.

A. *Workflow*

Figure 3 illustrates the overall workflow of I2PERCEPTION, which consists of five sequential phases. (i) *Low-cost RouterInfo collection* (§IV-B). We passively collect RouterInfo from all observable I2P routers using a small number of controlled floodfill routers, constructing a RouterInfo trace for each router. (ii) *Fine-grained router live behavior inference* (§IV-C). Using these RouterInfo traces, I2PERCEPTION infers each router's online sessions. We first derive coarse-grained sessions based on periodic RouterInfo publication, and then identify precise join and leave behaviors to obtain fine-grained online/offline intervals. (iii) *Online session complement* (§IV-D). Since low-cost data collection may miss certain RouterInfo, I2PERCEPTION applies a data-recovery–based complement mechanism to reconstruct incomplete sessions and restore missing join or leave behaviors. (iv) *Live behavior probing for the target hidden service* (§IV-E). In parallel, I2PERCEPTION actively probes the target hidden service to obtain its live behavior sequence, which reflects the availability of its hosting router. (v) *Live behavior correlation* (§IV-F). Finally, I2PERCEPTION computes the similarity between the hidden service's live behavior and that of all routers. The router with the highest behavioral consistency over time is identified as the hosting router.

B. *Low-cost RouterInfo Collection*

We collect *RouterInfos* from I2P routers by deploying only 15 controlled floodfill routers to participate in the netDB. Since I2P routers actively upload their latest *RouterInfo* to the netDB, we can passively gather this data from the controlled floodfill routers. Upon receiving *RouterInfos*, the routers extract and store the data fields into our database, allowing us to compile a series of *RouterInfo* traces for each router. We demonstrate the effectiveness of the low-cost collection approach using only 15 floodfill routers in Appendix A.

C. *Live Behavior Inference*

Our live behavior inference framework consists of three components: (i) **Online session inference**: We utilize the periodic status-maintenance mechanism of I2P routers, through which they generate and publish routine *RouterInfo*. By extracting all routine *RouterInfos* from the raw trace and analyzing the time intervals between them, we determine whether they are published within the same online session, enabling a coarse identification of each router's online and offline periods. (ii) **Join behavior identification**: We exploit the distinctive *RouterInfo* publication pattern exhibited by I2P routers during startup to identify join behavior at the beginning of each online session, thereby determining the router's precise join time. (iii) **Leave behavior identification**: For routers that publish a *RouterInfo* as part of their shutdown procedures, we identify these leave behavior *RouterInfos* at the end of each online session to determine the router's exact leave time. Table I summarizes the inherent *RouterInfo* management and publication mechanisms leveraged by each component, as well as how these mechanisms are exploited across different I2P router implementations.

TABLE I: Inherent mechanisms and exploitation approaches of each live behavior inference framework component.

Framework component	Router	Inherent mechanism	Exploitation approach
Online session inference	Java	Evaluate router status every 8–10 minutes, and forcibly publish a routine *RouterInfo* after every four evaluations.	Use a 32–43-minute interval to identify routine *RouterInfos* and divide sessions by how continuously they appear.
	C++	Evaluate congestion level every 12 minutes; update and republish a *RouterInfo* if the congestion flag changes.	Treat *RouterInfos* with congestion-flag changes as routine. Split online sessions when two consecutive routine *RouterInfos* are not separated by a multiple of 12 minutes.
Join behavior identification	Java	Upon startup, a router publishes an initial *RouterInfo*, which has a distinctive time gap (i.e., startup interval) from the first routine *RouterInfo*.	Using the startup interval, trace back from the first routine *RouterInfo* in an online session to find the initial *RouterInfo* and determine the join time.
	C++	At startup, the router initializes the congestion timer and issues its first *RouterInfo* after 0.5 seconds.	Trace back from the first routine *RouterInfo* to one published $12min \times n - 0.5s$ earlier, thereby identifying the router's join time.
Leave behavior identification	Java	Before going offline, a floodfill router publishes a *RouterInfo* announcing it is no longer floodfill.	Determine the router's leave time by detecting when its operation mode changes. (floodfill only)
	C++	Before going offline, the router publishes a *RouterInfo* with an 'E' congestion flag to prevent new connections.	Find the *RouterInfo* with the highest congestion level; its publication time is the router's leave time.

For clarity of exposition, we describe the fine-grained inference techniques using Java-based I2P routers below, as Java is the original implementation of I2P and most faithfully reflects the core design principles of the system. Due to space constraints, the corresponding technical details for C++-based routers are provided in the extended version of the paper available at [1]. The overall analytical structures for the two implementations remain the same, with differences arising primarily from implementation-specific publication schedules and event-driven behaviors.

Coarse-grained online session inference for Java-based routers. Java-based routers periodically execute an **update task** that checks their status (e.g., network congestion) and publishes a new *RouterInfo* if any status change is detected. The update interval, denoted as D_c, typically ranges from 8 to 10.5 minutes, calculated as follows:

$$D_c = \left(T \times \frac{3}{4} + random\left(S\right)\right) \div 4, \tag{2}$$

where T is a constant (default value of 43) and S is a random value between 0 and 10, to obfuscate the update time. Additionally, the execution of an update task is delayed until at least nine minutes have elapsed since the previous successful *RouterInfo* publication. Regardless of status changes, a router publishes a *RouterInfo* every fourth update task (approximately every 32 to 43 minutes, i.e., $4 \times D_c$). We refer to these periodic publication as **routine publish tasks**, and the data generated by them as **routine *RouterInfos***.

We identify a Java-based router's online sessions by analyzing its routine *RouterInfos*, distinguishable by specific publication intervals. If the interval between two *RouterInfo* publications equals four update intervals ($4 \times D_c$), such pairs are classified as routine *RouterInfos*. Starting from the first *RouterInfo* in a trace, we check if a subsequent publication appears within 32 to 43 minutes. If not, the first data point is not considered routine. This process continues until a pair of routine *RouterInfos* is found. Once identified, each routine *RouterInfo* serves as a baseline to locate the next, repeating the process until no further routine *RouterInfo* is found. All identified routine *RouterInfos* and the data between them are grouped into a single online session. We then resume the search from the *RouterInfo* following the last routine data to identify the next session using the same approach. Additionally, any non-routine *RouterInfo* between two sessions is assigned to the preceding session if its time gap from the session's last routine data is shorter than the routine interval.

Relying solely on routine *RouterInfos* may miss temporary offline behaviors due to random *RouterInfo* update intervals, which can vary by up to 10 minutes. As a result, *RouterInfo* from adjacent online sessions may overlap with routine intervals during temporary offline behavior, causing the coarse-grained method to misidentify session boundaries. This leads to two types of errors: **mismatch** and **misidentification** of routine *RouterInfo*. Mismatch errors occur when the interval between the last routine *RouterInfo* from the previous session (R_p^r) and a *RouterInfo* from the next session (R_s) coincides with the routine interval. This leads to two distinct results, depending on the characteristic of R_s: (1) If R_s is non-routine (**Case 1** in Table VI), R_s is mistakenly assigned to the previous session, causing incorrect inferences about the previous session's end time and the subsequent session's start time. (2) If R_s is routine (**Case 2** in Table VI), the two sessions are incorrectly merged, missing the offline period in between. Misidentification occurs when a non-routine *RouterInfo* after the last routine *RouterInfo* in the previous session (R_p) and R_s from the next session are mistakenly treated as successive routine publications. Since a floodfill router always sends a non-routine *RouterInfo* before it goes offline (we discuss it in our leave behavior identification method), this error generally occurs when dealing with floodfill routers, leading to two possible outcomes: (1) If R_s is also non-routine (**Case 3** in Table VI), R_p and R_s are incorrectly treated as part of the same session, splitting two sessions into three. (2) If R_s is routine (**Case 4** in Table VI), R_p is assigned to the next session, also causing incorrect inferences about the previous session's end and the subsequent session's start times. To address these challenges, we propose a fine-grained join behavior identification method for online sessions.

Join behavior identification for Java-based routers. To

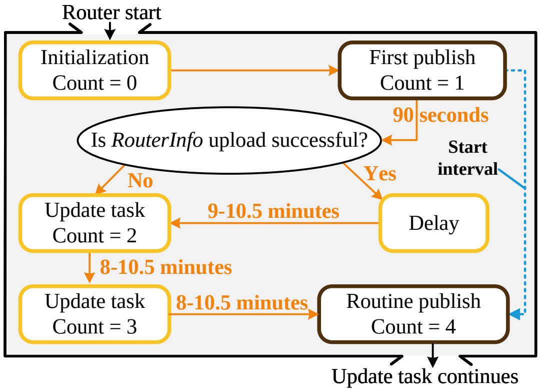


Fig. 4: Startup workflow of Java-based I2P routers.

enable fine-grained online session inference, we analyze the publication pattern of Java-based routers to identify their join behavior during startup. The routine *RouterInfo* is published every 4 update intervals, with a state variable C counting the tasks since startup. Specifically, *RouterInfo* is published when $C \bmod 4 = 0$. Upon startup, C starts at 0, and the first *RouterInfo* is published within 10 seconds, incrementing C to 1. The next task is scheduled 90 seconds later but cannot execute until 9 minutes after the latest upload. If the first upload succeeds within 90 seconds (i.e., the acknowledgment message for *RouterInfo* is received), the second task executes between 9 and 10.5 minutes, resulting in an interval of $3 \times D_c$ (25-31.5 minutes) between the first and first routine *RouterInfo*. Alternatively, if the router fails to receive the acknowledgment within 90 seconds, C increments to 2 after approximately 90 seconds. After two more update intervals, C reaches 4, triggering the first routine *RouterInfo*, with the interval between the first and first routine *RouterInfo* being approximately $90 + 2 \times D_c$ (17.5-22.5 minutes).

By analyzing the publication pattern during the startup phase of Java-based routers, we can accurately determine the start time of each online session. Starting from the first routine *RouterInfo* of an online session, we scan the entire trace for *RouterInfo* entries that match one of two possible join behaviors (i.e., a time interval of $3 \times D_c$ or $90 + 2 \times D_c$) to identify the first *RouterInfo* published after the router's startup. This approach allows for fine-grained identification of join behaviors, effectively addressing challenges posed by temporary offline behaviors.

While the join behavior identification method effectively corrects all the issues introduced by the coarse-grained method when dealing with the router's temporary offline behavior, it requires a complete join behavior—characterized by a distinct interval between two *RouterInfo* publications. If the router briefly returns online and publishes only one *RouterInfo*, the join behavior is incomplete (**Case 5** in Table VI), rendering the method inapplicable. In this case, we instead rely on the leave behavior identification method to correctly infer the end of the previous online session, and thereby infer the correct start time of the subsequent online session.

Leave behavior identification for Java-based routers. We find that only Java-based floodfill routers display explicit leave behavior. Immediately before shutdown, they publish a final *RouterInfo* whose *options* field removes the 'f' flag, indicating they will cease floodfill service. By scanning backward from the last routine *RouterInfo* in an online session, we locate this publication and use its timestamp as the router's leave time. This allows us to identify a router's offline behavior even if its join behavior is lost or absent in subsequent sessions. For non-floodfill routers, which lack explicit offline patterns, we estimate offline times to reduce discrepancies, as discussed in §IV-D.

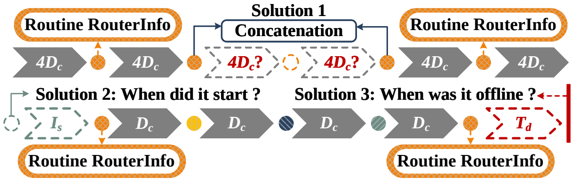


Fig. 5: Three solutions for online session complement.

D. Online Session Complement

While our live behavior inference method can theoretically group online sessions accurately, the low-cost deployment of floodfill routers leads to incomplete *RouterInfo* collection in practice. To address this issue, we verify the integrity of each identified session, detect missing join or leave behaviors, and supplement the missing information by analyzing the publication context. Although the analytical principles are consistent for both Java- and C++-based routers, the events that cause missing *RouterInfo* differ across implementations. To avoid redundancy, we present the Java-based techniques here and provide the corresponding C++-specific procedures in the extended version of this paper [1].

Missing *RouterInfo* can lead to three issues: loss of routine *RouterInfo* (**Cases 6–8 and 14** in Table VI), join behavior (**Cases 10–12 and 15**), and leave behavior (**Cases 9 and 16**), each causing errors in session inference. In the first issue, missing consecutive routine *RouterInfo* splits a session, leaving the second session without join behavior. In the second and third issues, missing data indicates the absence of join or leave behavior. We propose three solutions to address each issue and present how to handle online session inference errors in these situations, as shown in Fig. 5.

Solution 1: Online session concatenation. When join behavior is missing in both the first and second issues, we attempt to concatenate consecutive sessions to verify whether they belong to the same one (**Case 8** in Table VI). For Java-based routers, if the time interval between the final routine *RouterInfo* of the preceding session and the initial routine *RouterInfo* of the succeeding session matches a multiple of the routine interval, the sessions are merged. This condition can be expressed as:

$$\begin{cases} \min(D_c) \leqslant \frac{I_k}{m_t} \leqslant \max(D_c) \\ m_t = 4 \times k, \qquad\qquad k = 2, 3 \end{cases} \tag{3}$$

where I_k is the interval between two routine *RouterInfo*, and m_t is the number of update intervals within I_k. We limit I_k to a maximum of three routine intervals, as empirical testing shows no instance of consecutively losing more than two routine *RouterInfo*.

Solution 2: Join behavior supplementation. If a session has no join behavior and cannot be merged using **Solution 1**, the join behavior is considered missing or incomplete. For Java-based routers, the join behavior consists of two *RouterInfos*. When one is missing, we determine which publication is absent and infer its expected position based on the characteristic startup interval. The initial *RouterInfo* should be published an interval before routine behavior begins, referred to as the *startup interval*, measured as either $3 \times D_c$ or $90 + 2 \times D_c$, denoted by D_{s1} and D_{s2}, respectively. If the first routine *RouterInfo* is missing (**Case 6** in Table VI), the expected gap between the initial and the next routine publication becomes the sum of the startup and routine intervals—either $7 \times D_c$ or $90 + 6 \times D_c$. We thus search backward from the first captured routine *RouterInfo* within this time window to locate a potential initial *RouterInfo*. If found, its publication time is regarded as the session's start time. If not, we infer that the missing data is the initial *RouterInfo* (**Case 10** in Table VI). In this case, we compute the expected values for D_{s1} and D_{s2} as follows:

$$\begin{cases} E(D_{s1}) = 90 + 2 \times E(D_c) \\ E(D_{s2}) = 3 \times E(D_c) \end{cases} \tag{4}$$

where $E(D_c)$ is the expected duration of an update interval, given by $\frac{3T+4E(S)}{16}$. According to Equation (2), and given that the probabilities of these two cases are generally equal, the expected value of the startup interval, E_f, is calculated as

$$E_f = \frac{E(D_{s1})}{2} + \frac{E(D_{s2})}{2} = 45 + \frac{15T + 20E(S)}{32} \tag{5}$$

Thus, the start time of the online session can be estimated as E_f earlier than the publication time of the first routine *RouterInfo*.

Solution 3: Leave behavior supplementation. Our goal is to estimate the expected remaining online duration after the router's final observed *RouterInfo* (**Case 9** and **16** in Table VI). For Java-based routers, non-floodfill routers do not exhibit explicit leave behavior, while floodfill routers may lose the *RouterInfo* without a 'f' flag, which serves as an indicator of leave behavior. In both cases, since no additional routine *RouterInfo* is received after the router's last *RouterInfo*, we can approximate that the router goes offline before the next routine *RouterInfo* would appear. We first estimate the expected publication time for the next routine *RouterInfo* T_n by:

$$T_n = T_r + 4 \times E(D_c) \tag{6}$$

where T_r denotes the publication time of the last routine *RouterInfo*. Then the inferred offline time T_o is given by

$$T_o = \frac{T_n - T_f}{2} + T_f = \frac{T_r + T_f}{2} + 2E(D_c) \tag{7}$$

where T_f represents the publication time of the last *RouterInfo* in the online session.

Discussion: We analyze the remaining cases (**Cases 7, 11, and 12** in Table VI) caused by missing *RouterInfo* that cannot be resolved using the proposed solutions. **Case 7** occurs when the last routine *RouterInfo* for a non-floodfill router is lost. Without explicit leave behavior, this results in insufficient timing information to infer the router's leave time, leading to an inference bias of about 35–45 minutes. **Case 11** arises when the initial *RouterInfo* for a Java-based non-floodfill router is lost after a brief offline period, causing an incomplete join behavior. If the interval between the first routine *RouterInfo* of the subsequent session (R^s_f) and the last routine data (R^p_ℓ) of the previous session equals the sum of the startup and routine intervals, our **Solution 2** may incorrectly infer that a routine *RouterInfo* between them is lost, treating R^p_ℓ as the initial *RouterInfo* of the current session. Although this error cannot be corrected without both join and leave behaviors, the two sessions remain distinguishable, and both inference biases of the previous session's end time and the subsequent session's start time are within a routine interval (less than 40 minutes). If the interval between R^s_f and R^p_ℓ coincides with the routine interval, the coarse-grained method may incorrectly merge the sessions (**Case 12**). Although uncorrectable, this case is rare due to the specific timing requirement.

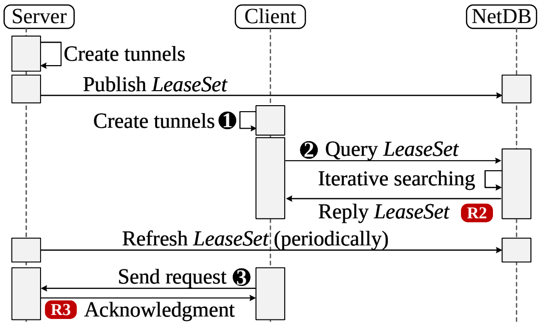


Fig. 6: Process of accessing a hidden service.

E. Live Behavior Probing for Hidden Service

Unlike the passive method for routers, we use an active probing approach to infer the live behavior of a target hidden service. As shown in Fig. 6, the process for an I2P client to access a hidden service involves three stages: (i) the client establishes a pair of unidirectional tunnels for communication. (ii) The client queries the netDB for the target hidden service's LeaseSet. If the response message $R2$ includes the *LeaseSet*, the lookup is successful; otherwise, it indicates that the *LeaseSet* is not present in the netDB. (iii) Using the *LeaseSet*, the client connects to the hidden service as depicted in §II-B and sends a service request, receiving acknowledgment via message $R3$.

Based on this mechanism, the hidden service's live behavior can be inferred from the messages at each stage. If a failure response $R2$ is received, it indicates the hidden service has been offline for more than 10 minutes, causing the *LeaseSet* to expire. If the *LeaseSet* is retrieved but no connection is made, the service is inferred to have been offline within the last 10 minutes. Successfully receiving $R3$ confirms the service's availability. By periodically repeating this access task, the live behavior of the hidden service can be inferred. Due to the I2P network's inherent anonymity, our client remains unexposed to the hidden service. In addition, I2P reliably delivers probing messages to the target service by using the SSU (Secure Semireliable UDP) and NTCP (NIO-based TCP) transport protocols, as well as the application-layer streaming library.

The network environment has limited impacts on the attack's effectiveness. I2P provides approaches to ensure the reliability of data transmission. At the transport layer, I2P implements the SSU and NTCP protocols based on UDP and TCP, respectively, to ensure reliable message delivery. At the application layer, I2P provides the streaming library for the same purpose. Because of the above strategies, as long as the host router's RouterInfo and the hidden service's response to probing by the attacker could be delivered correctly, our attack can succeed.

F. Live Behavior Correlation

We propose a dynamic time warping (DTW)-based correlation method to evaluate the similarity between the live behaviors of routers and a target hidden service. First, we serialize the inferred live behavior by representing online and offline states with positive and negative integers, respectively, while recording the duration of each state. For example, if a router stays online for 5 minutes and then offline for 5 minutes, its state sequence would be: 1, 2, 3, 4, 5, -1, -2, -3, -4, -5. The DTW algorithm then computes the alignment cost between two sequences by calculating the pairwise distance between corresponding elements. Unlike traditional methods that rely on predefined distance metrics (e.g., Euclidean distance), we use a customized distance function: if two numbers are identical, the distance is 0; if they share the same sign, the distance is 1; and if they have opposite signs, the distance is 2. The DTW algorithm outputs a distance score, where a lower score indicates higher similarity between the two live behaviors.

G. Theoretical Analysis

We now prove that a target hidden service exhibits a unique live behavior pattern over time and can therefore be uniquely identified if this behavior is measurable, as demonstrated in this paper. We can model the live behavior of a hidden service as an on-off stochastic process. Therefore, n hidden services correspond to n on-off stochastic processes. Assume that their *on* periods follow the same discrete probability distribution and have k values with probabilities $a_1, \dots, a_k$. Their off periods follow the same discrete probability distribution and have l values with probabilities $b_1, \dots, b_l$. While one might assume the n hidden services are more distinct, this model assumes they are more similar, presenting a worse-case scenario for the attacker. Assume that the n processes are independent and their on-off cycles are independent, where one cycle contains one *on* period and one off period.

Theorem 1: The probability $P_{\mathcal{A}}$ that one particular on-off process A does not produce the same length-m on-off sequence with any of the other on-off $n-1$ processes can be calculated as follows,

$$P_{\mathcal{A}} = \left(1 - (S_a S_b)^m\right)^{n-1}, \tag{8}$$

where

$$S_a = \sum_{i=1}^{k} a_i^2, \qquad S_b = \sum_{j=1}^{l} b_j^2. \tag{9}$$

The detailed derivation can be found in Appendix B. It can be observed that in Equation (8), $S_a S_b < 1$. Therefore, when m is large, $P_{\mathcal{A}}$ approaches 1. That is, we can uniquely identify a hidden service given a long observation of its live behavior.

Corollary 1.1: If $\mathcal{A}$ and the other on–off processes are synchronized in their on–off behavior, they become identical and cannot be distinguished based solely on that behavior.

V. Evaluation

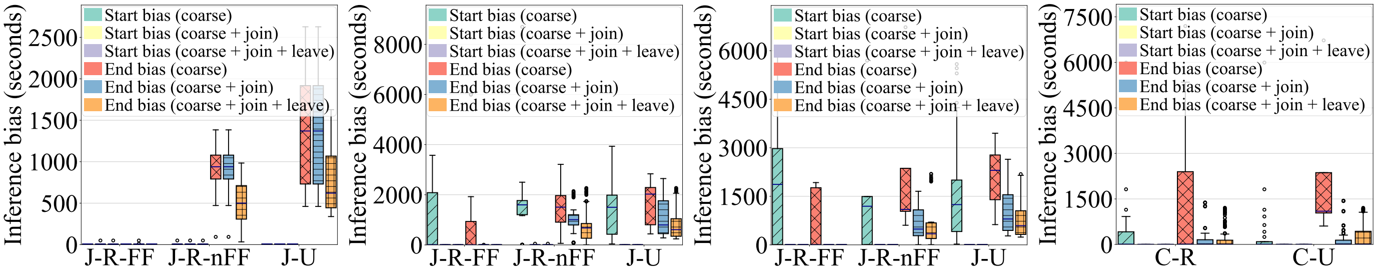
We implemented I2PERCEPTION and conducted extensive experiments within the real-world I2P network under controlled conditions. In this section, we first present details of our experimental setup and then present the experimental results, addressing three research questions as follows:

(RQ1): Are the inferred live behaviors of I2P routers consistent with their actual behaviors using *RouterInfo* data (§V-B1)?
(RQ2): Can our method accurately reconstruct the router's live behavior when we have incomplete data (§V-B2)?
(RQ3): Can I2PERCEPTION effectively deanonymize the target hidden service (§V-B3)?

A. Experiment Setup

Experiment environment. We deploy 25 I2P routers on virtual private servers (VPS) running Ubuntu 22.04, distributed across various regions. Each VPS is configured with 1 virtual CPU, 1 GB RAM, and 32 GB storage. All routers use customized I2P software to collect *RouterInfo* data and log essential information without disrupting normal network operations. 15 routers operate in floodfill mode to support large-scale *RouterInfo* collection, with the sufficiency of 15 floodfill routers demonstrated in Appendix A. The remaining 10 routers host the hidden services used as targets in our deanonymization experiments: 5 Java-based and 5 C++-based. Among the Java-based routers, 3 have reachable ports and two are firewalled. One of the reachable Java routers is configured in the floodfill mode to validate our leave behavior identification method. All five C++-based routers are configured in the non-floodfill mode, with 3 having reachable ports and 2 firewalled. An I2P client is deployed on a Windows 11 machine (Intel Core i7-1165G7, 16 GB RAM) to probe the controlled hidden services and process data for the deanonymization prototype.

Dataset Collection. We collect *RouterInfo* data from 15 controlled floodfill routers over a period of more than 8 months. To create a *ground truth dataset*, we locally record the *RouterInfo* publication logs from 10 host routers, resulting in a complete, lossless dataset. This ground truth dataset serves to evaluate the contribution of each component in the fine-grained inference method and to validate the accuracy of inferred live behavior under ideal conditions. Additionally, we capture *RouterInfos* from the 15 deployed floodfill routers to build an *evaluation dataset*, which may experience partial data loss due to network dynamics. This dataset is used to assess both the necessity of the online session complement strategy and the overall effectiveness of the live behavior inference framework under realistic conditions. These two datasets allow us to evaluate the performance of our inference method in both ideal (no data loss) and realistic (with data loss) settings.



(a) (b) (c) (d)

Fig. 7: Results of the ablation experiments. Subfigures (a–c) show results for Java-based routers under different offline durations: (a) 45–60 min (S1), (b) 30–45 min (S2), and (c) 5–10 min (S3). Subfigure (d) presents results for C++-based routers with mixed offline durations (S4). J-R-FF, J-R-nFF, and J-U denote Java-based floodfill, non-floodfill, and unreachable-port routers, respectively. C-R and C-U refer to C++-based routers with reachable and unreachable ports.

Controlled Online/Offline Behavior Scenarios. We configure the online/offline behaviors of the controlled host routers and consider seven scenarios as shown in Table II to evaluate the accuracy of our live behavior inference method. Router behaviors are controlled using *crontab* scripts to follow the specified configurations automatically. **S1** to **S4** represent hidden services that exhibit frequent on–off patterns, posing challenges for inferring the live behavior of their host routers, specifically, accurately mapping each *RouterInfo* to its associated online session. Services in **S1** to **S4** run for two months. Their daily behavioral patterns are defined in Table II. Positive values indicate the duration of online sessions, while negative values represent offline periods, with their absolute values denoting the length. **S5** to **S7** correspond to long-lived hidden services, with the hidden services in **S5** to **S7** run for the number of days specified in Table II.

For scenarios of hidden services with frequent on-off behaviors, **S1** to **S3** target the services hosted on Java-based routers. **S1** tests the coarse-grained method's ability to segment online sessions, with routers remaining offline for 45–60 minutes, longer than the routine publication interval but shorter than the *RouterInfo* expiration, causing a false online identification by floodfill routers. **S2** introduces two types of errors (routine *RouterInfo* mismatches and misidentifications) to evaluate the effectiveness of the join behavior identification method, with offline durations set to 30–45 minutes, aligning with the routine publication interval. **S3** evaluates the leave behavior identification method by introducing incomplete join behaviors, with offline durations reduced to 5–10 minutes, increasing the likelihood of mistakenly merging sessions due to a short offline period. **S4** focuses on the services hosted on C++-based routers, where the offline duration has a minimal impact on the coarse-grained method. This is due to the lack of randomness in routine publication intervals, which eliminates the mismatch and misidentification issues commonly encountered with Java-based routers. Thus, we randomly select offline durations between 5 and 60 minutes to create a varied environment for evaluating the inference method.

TABLE II: Live behavior settings for different scenarios.

Hidden services with frequent on-off behaviors (Time values are in minutes)	
S1	100, -45, 110, -50, 120, -55, 130, -60, 100, -55, 110, -50, 120, -45, 130, -160
S2	120, -30, 110, -35, 100, -40, 120, -45, 110, -40, 100, -30, 10, -35, 120, -30, 10, -35, 100, -40, 120, -60
S3	140, -10, 130, -5, 120, -10, 110, -5, 10, -10, 100, -10, 130, -5, 10, -10, 120, -10, 110, -5, 100, -10, 110, -5, 10, -10, 120, -15
S4	160, -20, 160, -15, 160, -10, 10, -5, 160, -30, 160, -40, 10, -10, 160, -50, 160, -60, 10, -50
Hidden services with stable live behaviors	
S5	Online for nine days; Offline on Day 10
S6	Online for 28 days; Offline on Day 29
S7	Online for 50 days; Offline on Day 51

For scenarios of long-lived hidden services, in **S5**, a hidden service stays online for nine days and goes offline on Day 10. In **S6**, a hidden service remains online for 28 days before going offline on Day 29. In **S7**, a hidden service runs continuously for 50 days before shutting down on Day 51.

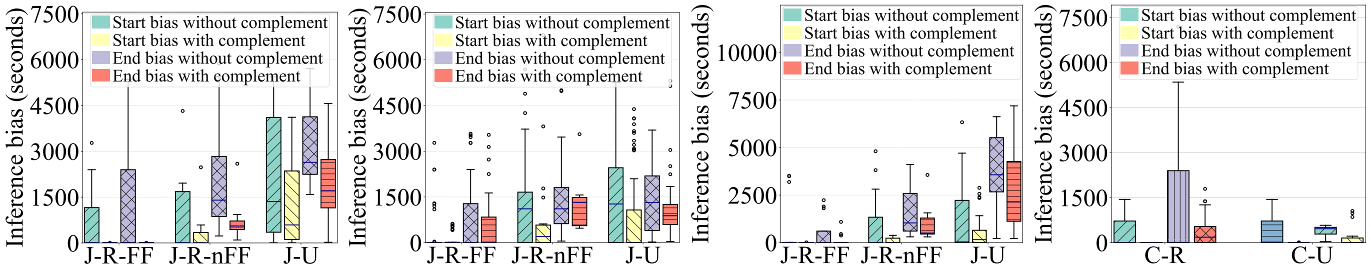
B. Experiment Results

1) Consistency (RQ1)

TABLE III: Occurrence of each case in Table VI, triggered by different types of routers under various scenarios. In S1-S3, R-FF, R-nFF, and U denote floodfill, non-floodfill, and unreachable-port routers, respectively. In S4, R and U refer to C++-based routers with reachable and unreachable ports.

	S1: 45-60 (J)			**S2: 30-45 (J)**			**S3: 5-10 (J)**			**S4: MIX (C)**	
	R-FF	R-nFF	U	R-FF	R-nFF	U	R-FF	R-nFF	U	R	U
Case 1	-	-	-	77	152	228	76	86	256	-	-
Case 2	-	-	-	19	22	41	117	137	178	-	-
Case 3	-	-	-	252	-	-	-	-	-	-	-
Case 4	-	-	-	-	-	-	161	-	-	-	-
Case 5	-	-	-	25	23	42	67	67	68	-	-
Case 13	-	-	-	-	-	-	-	-	-	51	65

In the experimental scenarios we constructed, along with the corresponding live behaviors assigned to routers, all cases challenging the effectiveness and reliability of the coarse-grained method occur. Table III reports the occurrences of routine *RouterInfo* mismatches (Cases 1 and 2), misidentifications (Cases 3 and 4), and incomplete join behaviors (Cases 5 and 13), underscoring the need for our proposed join and leave behavior identification methods.



(a) (b) (c) (d)

Fig. 8: Comparison of live behavior inference biases with and without the online session complement method, using the *evaluation dataset*. Subfigures (a–d) show results under four offline duration settings: (a) S1 (45–60 min), (b) S2 (30–45 min), (c) S3 (5–10 min), and (d) S4 (mixed). Router categories and scenario definitions follow those in Fig. 7.

We conduct ablation experiments to evaluate the necessity of each component in our I2P router live behavior inference framework. Beginning with the coarse-grained method, we incrementally incorporate join and leave behavior identification, as well as the leave behavior complement method, to achieve fine-grained inference. Accuracy is measured by the absolute time difference (in seconds) between actual and inferred join and leave times. Results for all four controlled scenarios of **S1**–**S4** are presented in Fig. 7.

Fig. 7a shows that in **S1**, the coarse-grained method accurately infers all online session start times, with a median bias of just 6 seconds. However, the accuracy of leave time inference varies across router settings. Since all *RouterInfo* are correctly assigned to their respective sessions and the first *RouterInfo* of each session is accurately identified, the join behavior identification method offers no further improvement. For floodfill routers, the coarse-grained method achieves high accuracy in leave time inference, with a median bias below 10 seconds, as it captures all *RouterInfo* published after the last routine *RouterInfo* within the same session. In contrast, for non-floodfill routers lacking explicit leave behavior, the median leave time bias reaches approximately 900 seconds, which can be reduced to around 500 seconds using the leave behavior complement method.

Fig. 7b and Fig. 7c show that the coarse-grained method exhibits significant biases in inferring the join times of controlled routers in scenarios **S2** and **S3**, with several exceeding 2000 seconds, despite a median below 10 seconds. After incorporating the join behavior identification method, these biases are dramatically reduced to less than 10 seconds. Likewise, applying the leave behavior complement method to non-floodfill routers cuts the leave time bias by more than half, lowering the median from 1400 seconds to below 700 seconds (12 minutes), which has minimal impact on the similarity between inferred and actual live behaviors. These results confirm the fine-grained method's effectiveness in enhancing inference accuracy.

Fig. 7d presents the live behavior inference results for C++-based routers in **S4**. The coarse-grained method, limited to approximating session boundaries, yields substantial inference biases. Incorporating the join and leave behavior identification methods significantly improves accuracy, reducing the median join time bias to under 5 seconds and the median leave time bias to below 10 seconds. Although observable leave behavior occasionally fails to appear—resulting in a few outliers—the leave behavior complement method effectively bounds all leave time biases within 15 minutes.

The above results indicate that when we have complete data, our live behavior inference method maintains a high level of consistency with the ground truth across various scenarios.

2) Reliability (RQ2)

Table IV presents the occurrences of incorrect inferences in the fine-grained online session inference method caused by missing *RouterInfo*, as listed in Table VI, using the evaluation dataset collected from controlled floodfill routers. All cases occur, though with varying frequencies. Cases 6–10, and 14–16 appear more often and are successfully corrected by our online session complement methods. We further analyze the two unresolvable cases: 11 and 12. **Case 11** is observed to be extremely rare. Although it could theoretically arise in **S2** for non-floodfill routers with reachable ports, no such instance is observed in practice. Similarly, while **Case 12** may substantially affect inference accuracy, its occurrence is also, as expected, exceedingly rare. Then we quantify the effectiveness of our online session complement method by comparing the accuracy of inferred join and leave times with and without the session complement method.

To evaluate the effectiveness of the online session complement method, we quantitatively compare the live behavior inference biases across different router types and scenarios,

TABLE IV: Occurrence of each case in Table VI, triggered by different types of routers under various scenarios.

	S1: 45-60 (J)			**S2: 30-45 (J)**			**S3: 5-10 (J)**			**S4: MIX (C)**	
	R-FF	R-nFF	U	R-FF	R-nFF	U	R-FF	R-nFF	U	R	U
Case 6	-	-	-	-	13	21	-	37	55	-	-
Case 7	-	35	28	-	22	16	-	25	14	-	-
Case 8	33	49	57	27	43	51	48	45	59	-	-
Case 9	11	-	-	17	-	-	19	-	-	-	-
Case 10	11	10	18	8	19	33	7	15	26	-	-
Case 11	-	-	-	-	0	0	-	5	12	-	-
Case 12	-	-	-	-	3	3	-	2	7	-	-
Case 14	-	-	-	-	-	-	-	-	-	26	21
Case 15	-	-	-	-	-	-	-	-	-	47	51
Case 16	-	-	-	-	-	-	-	-	-	31	22

TABLE V: Upper quartile value of inference biases in join and leave times for various router categories, with and without session complement method, and the corresponding similarity distance to the ground truth per hidden service's online session.

Complement	J-R-FF			J-R-nFF			J-U			C-R			C-U		
	join	leave	τ	join	leave	τ	join	leave	τ	join	leave	τ	join	leave	τ
✗	353	1147	25	1677	2336	66	2652	3722	106	868	2380	54	867	542	23
✓	6	606	10	361	702	18	561	1436	33	2	430	7	3	162	3

using the fine-grained inference method alone and with the complement method, based on the evaluation dataset. The results are presented in Fig. 8.

Fig. 8a shows the inference biases of controlled host routers' join and leave times in **S1**. Java-based floodfill routers, which exhibit explicit join and leave behaviors, are minimally affected by missing *RouterInfo*, with median inference biases for both join and leave times below 10 seconds. In contrast, Java-based non-floodfill routers are more vulnerable to data loss. When using only the fine-grained method, the median leave time bias reaches 1400 seconds, while the online session complement method reduces it to 600 seconds. For Java-based unreachable routers, limited connectivity hampers *RouterInfo* propagation compared to reachable routers, resulting in more missing *RouterInfos*. Consequently, the fine-grained method produces median biases of 1350 seconds for join times and 2600 seconds for leave times, which the complement method effectively reduces to 600 and 1700 seconds, respectively.

Fig. 8b presents the performance of the online session complement method in **S2**. The method significantly improves inference accuracy for Java-based floodfill routers, achieving median biases of less than 10 seconds for both join and leave times. For non-floodfill routers, since multiple cases occur more frequently than in floodfill routers (as shown in Table IV), larger inference errors are observed. Regarding routers' join times, the online session complement method decreases the median bias from 1130 seconds to 230 seconds. For leave times, although the complement method slightly raises the median bias, it effectively eliminates most outliers by correcting the majority of online session misidentifications, thereby concentrating the inference biases within a lower range of 600–1500 seconds. Finally, for unreachable routers, the online session complement method is also highly effective, reducing the median inference biases for join times from 1250 seconds to within 10 seconds, and for leave times from 1300 to 800 seconds.

Fig. 8c shows that, in **S3**, the online session complement approach significantly enhances the accuracy of join and leave time inference for Java-based routers. Along with the results in Fig. 8a and Fig. 8b, these findings demonstrate the robustness of I2PERCEPTION in accurately inferring the live behaviors of Java-based routers under real-world conditions.

Fig. 8d illustrates the performance of our session-complement approach for C++-based routers in **S4**. For reachable routers, the method reduces all join time inference biases to within 10 seconds and eliminates most inference biases when inferring leave times, keeping 75% of values below 550 seconds. For unreachable routers, join time inference accuracy is comparable to that of reachable ones. The leave time inference shows even greater improvement, with nearly all inference biases constrained within 240 seconds.

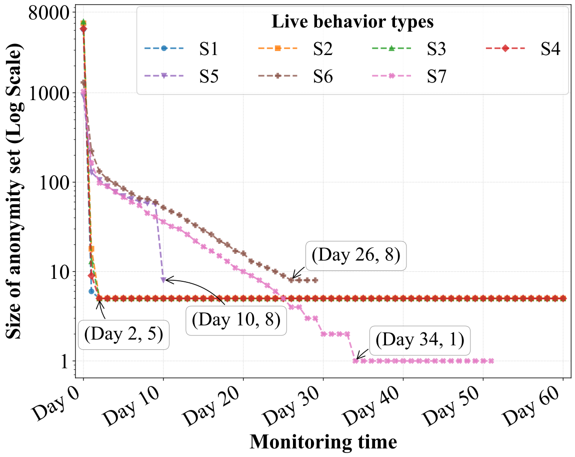


Fig. 9: Hidden services' anonymity set size vs. time. Note: When a scenario contains only a single service and no other synchronized services, that service can be uniquely identified: S1, S2, S3, and S4 on Day 2; S5 on Day 10; S6 on Day 26; and S7 on Day 34.

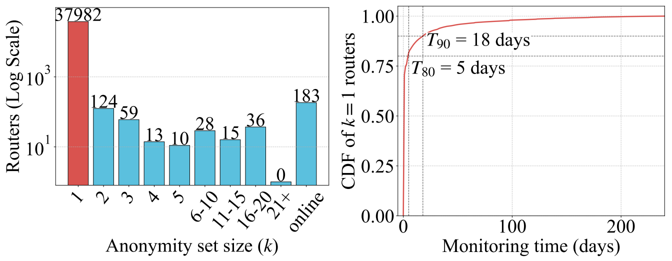
Table V summarizes the inference accuracy of join and leave times for different types of I2P routers across all scenarios, comparing the results of using the fine-grained online session inference method alone and in combination with the online session complement method. In the table, each 'join' column represents the upper quartile of biases (i.e., 75% of biases are less than or equal to this value) for all inferred join times of the corresponding router type. Likewise, the "leave" column indicates the upper quartile of leave time inference biases. These results highlight that the session complement method substantially enhances the inference accuracy of router live behaviors in real-world conditions.

Based on the join and leave time inference biases discussed above, we derive the similarity thresholds used to partition anonymity sets in deanonymization attacks against hidden services in the real-world environment. The experimental results indicate that inference biases vary across different router types. Accordingly, for each online session (i.e., one on–off cycle) of a target hidden service, we assign distinct similarity thresholds to different router categories to determine whether a router belongs to the service's anonymity set. Thr_{DTW} in Table V denotes the per-session similarity threshold for each router type, derived from its corresponding live behavior inference accuracy. Consequently, the overall threshold for determining a router's inclusion in the anonymity set is defined as $n \times thr_c$, where n is the number of the hidden service's online sessions and thr_c is the category-specific threshold.

3) Effectiveness (RQ3)

To evaluate the real-world performance of I2PERCEPTION, we conduct deanonymization attacks on controlled hidden services under the scenarios defined in Table II. For each scenario **S1–S4**, five hidden services are configured to follow the same live behavior described in §V-A. For the long-lived scenarios **S5** and **S6**, eight services are deployed for each configuration, while one service is run for **S7**. Throughout their runtime, we continuously monitor the I2P network and record the size of each hidden service's anonymity set at the end of each day. Please note: In practice, it is unlikely for two hidden services to exhibit the same on-off pattern, and a real-world hidden service will be uniquely identified. We create synchronized processes to illustrate Corollary 1.1.

Fig. 9 presents how the anonymity sets of controlled hidden services shrink over time. For the short-lived hidden services (**S1–S4**), their anonymity sets rapidly shrink from 5,000-6,000 I2P routers to approximately 10 on the first day. By day 2 and onward, their anonymity sets shrink to 5, corresponding to the number of synchronized services. In the long-lived scenarios **S5–S7**, 915, 1,307, and 1,035 I2P routers go online around the same time. These routers form their initial anonymity sets. After one day, these sets shrink to 129, 223, and 164 routers. In **S5**, the set further decreases to 57 routers by day 9 and to 8 by day 10, corresponding to the number of synchronized services. In **S6**, routers in the initial set gradually go offline, reducing the anonymity set to 8 by day 26. Finally, the single hidden service in **S7** is successfully deanonymized after 34 days of monitoring.



(a) (b)

Fig. 10: Experimental results from long-term monitoring of the I2P network. (a) Number of routers with different anonymity set sizes (k). (b) CDF of the number of routers exhibiting unique behaviors over the monitoring period.

We also evaluate the behavioral distinguishability of real-world I2P routers by examining how many exhibit on–off patterns that are unique with respect to our live behavior inference framework and a predefined similarity threshold (Refer to Appendix C for the choice of the threshold). We include all routers that appeared online on the first monitoring day and track their behaviors for the subsequent eight months. Each day, we compute the similarity distance between every router and all others to derive its anonymity set. A router is classified as behaviorally unique once no other router remains in its anonymity set. Fig. 10a summarizes the results. Among all monitored routers, 37,982 eventually exhibit unique behaviors, while 468 share a similar behavior with at least one other router. Fig. 10b further shows how quickly uniqueness emerges: 90% of the unique routers become distinguishable within the first 18 days. These findings highlight the substantial behavioral diversity among I2P users, which implies that hidden services hosted on such routers may produce similarly distinguishable behavioral patterns, potentially increasing their deanonymization risk. For routers without unique behaviors, our method may not deanonymize them immediately, but it significantly reduces their anonymity set from thousands to tens or hundreds, enabling strong adversaries to focus active attacks (e.g., DoS, watermarking, fingerprinting) on a much smaller candidate pool.

VI. Mitigation

To ensure a comprehensive treatment of the security implications of our findings, we also analyze potential countermeasures to our attack. Mitigation can be considered from two complementary perspectives: **(I) protocol-level defenses** that reduce information leakage from RouterInfo publication, and **(II) user-side measures** that make hidden services harder to correlate.

Protocol-Level Defenses. We propose two protocol-level strategies. **(a) Randomizing Publication Intervals:** We can enhance the randomness of *RouterInfo* publication by substituting the fixed four update interval publication rule (as introduced in §IV-C) with a randomized mechanism that sets the next mandatory publication within 10–55 minutes after each one. This broad randomization window prevents attackers from identifying online sessions through periodic patterns and, in real-world environments, removes the analytical basis for handling uncaptured *RouterInfo*. Importantly, this approach still guarantees timely *RouterInfo* refreshing in the netDB, since the maximum publication interval remains shorter than the *RouterInfo* timeout threshold (i.e., one hour),making this mitigation strategy both secure and operationally adaptable. **(b) Eliminate Join and Leave behaviors:** Eliminating special *RouterInfo* publication patterns during router initialization and shutdown phases can prevent attackers from precisely identifying the start and end of online sessions. Also, without these behaviors, a real-world attacker cannot determine whether the first *RouterInfo* within a *RouterInfo* trace is the initial one published after a router starts or if previous *RouterInfos* were lost. This makes it infeasible to recover complete online sessions when only partial *RouterInfo* data is available.

Real-World Deployment by the I2P Team. Following our responsible disclosure, I2P provides a patch to mitigate the anonymity risk from three perspectives. First, to introduce randomization, the patch dictates that each routine *RouterInfo* publication has a 1-in-32 chance of being skipped. Second, the patch refines the management of the update counter and removes the specific code branch executed for the initial *RouterInfo* publication, effectively eliminating the 'join behavior' outlined in §IV-C. Finally, a floodfill router now does not publish a non-floodfill *RouterInfo* during a non-graceful shutdown, thus removing the 'leave behavior'.

We also propose two user-side measures: **(a) Synchronizing the behavior of a group of I2P hidden services.** By coordinating several hidden services to exhibit identical behaviors, the host routers of these services can collectively form a larger anonymity set, preventing attackers from associating a single hidden service with a unique router. Although this strategy can effectively expand the anonymity set, it is generally impractical because hidden services operated by different users typically act independently. **(b) Detecting active probing of I2P services.** Although the hidden service cannot observe a client's IP address, it can observe the client's I2P identity. Repeated or periodic probing from the same identity, or probing with distinguishable timing patterns from different sources may reveal attempts to infer live behavior, allowing the service to ignore or throttle such requests.

VII. Related Work

Over the years, numerous anonymous communication systems have been developed, such as Tor [3], I2P [48], to provide users with anonymous access to Internet services. Meanwhile, various attacks have been investigated to break the users' anonymity. Broadly speaking, the core idea of deanonymization is to exploit identifiable behavioral patterns over time to gradually reduce the anonymity set of a target user, ultimately isolating the target from a crowd of users.

In the I2P network, researchers have applied deanonymization strategies by leveraging the network's unique protocol behaviors and inherent vulnerabilities. Herrmann *et al.* [49] attempted to trace hidden services by correlating traffic observed on controlled routers participating in the service's tunnels. Other attacks have targeted specific protocol vulnerabilities. Egger *et al.* [50], for instance, launched a Sybil attack against hidden services. Their method involved using numerous floodfill routers to intercept the published *RouterInfo* of a service's host router, thereby revealing the identity of the associated tunnel's exit router. In another approach, Crenshaw *et al.* [51] proposed a side-channel method that correlates anonymous I2P websites (eepsites) with their public-facing counterparts on the clearnet to uncover the server's true IP address. Although prior studies have demonstrated successful deanonymization, they either exploited protocol vulnerabilities in I2P or relied on traffic-correlation techniques. The protocol vulnerabilities leveraged by previous work have been patched and are no longer exploitable. Moreover, traffic-correlation attacks require the adversary to occupy specific positions within a hidden service's client tunnel, which is extremely difficult to achieve in the I2P network at its current scale.

More recently, research has shifted toward analyzing behavioral patterns of I2P routers. Simioni *et al.* [52] inferred a router's online and offline status based on the duration its *RouterInfo* remained in the netDB, thereby identifying behavioral differences among routers. However, their work does not uncover the *RouterInfo* publication mechanism and, as a result, cannot achieve high accuracy when inferring I2P routers' live behaviors. Moreover, in real-world scenarios where capturing all *RouterInfos* is impossible, their method cannot handle the impact of *RouterInfo* loss, thus making it unable to achieve reliable deanonymization.

Previous researchers on Tor have adopted different types of identifiable patterns, ranging from embedding actively crafted traffic features into flows of anonymous systems [4]–[12], mining intrinsic traffic characteristics of anonymity systems [13]–[35], detecting side-channel behaviors such as clock skew on the host of a target hidden service [53], to identifying protocol flaws that cause behavioral deviations across anonymous users [37]. Existing deanonymization methods for Tor hidden services do not leverage user-driven behavioral patterns that produce distinctive on–off activity in hidden services.

VIII. Conclusion

We propose I2PERCEPTION, a de-anonymization framework for I2P hidden service. Using the routers' live behaviors inferred by a passive monitoring approach to correlate with the service's live behavior obtained via an active probing approach, we may identify the server's host router to reveal the IP address of the hidden server. To precisely infer the I2P router's fine-grained live behavior, we explore the publication rules of *RouterInfo* data during join and leave period to precisely identify the router's join and leave behavior. Also, we investigate the online session recovery method to reduce the impact of the incomplete data caused by low-cost data collection strategy. The real-world experiment results shows that the live behavior inferred by I2PERCEPTION has high consistency with the router's actual live behavior. Moreover, all the hidden services we deployed were successfully deanonymized by I2PERCEPTION.

Acknowledgment

We thank the anonymous reviewers for their constructive comments and suggestions. This work is supported in part by the National Natural Science Foundation of China (NSFC) under Grant Nos. 62232004, 92467205 and 62202099, the Natural Science Foundation of Jiangsu Province under Grant No. BK20220806, the 'Zhishan' Young Scholar Program of Southeast University under Grant No. 2242024RCB0012, the Jiangsu Provincial Key Laboratory of Network and Information Security under Grant No. BM2003201, the Key Laboratory of Computer Network and Information Integration of Ministry of Education of China under Grant No. 93K-9, and the Collaborative Innovation Center of Novel Software Technology and Industrialization. We also acknowledge the support of the Big Data Computing Center of Southeast University. Any opinions, findings, conclusions, and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

Ethics Considerations

In this paper, we collected I2P router's *RouterInfo* data, and periodically sniffed several controlled I2P hidden services, aiming to correlate the live behavior of the target hidden services with their host routers, so as to achieve de-anonymization. We took the following steps to ensure that our experiments were conducted ethically:

Responsible Disclosure: Upon discovering the vulnerabilities, we disclosed our findings to the I2P maintainers to support timely mitigation. Our work was acknowledged by the I2P project, and as of this submission, the reported issues have been addressed in a prior I2P release. [44].

Controlled Deployment: All hidden services considered as attack targets are deployed on controlled routers by ourselves, and the identities of these services are not disclosed to anyone, ensuring that no user could accidentally access these hidden services.

I2P Academic Research Guidance: Our experiments strictly adhere to the security recommendations on the I2P Academic Research website [54]. Specifically, we ensure the following: no active exploits or Denial of Service (DoS) attacks are performed on the I2P network; no social engineering is conducted targeting I2P team members or community participants; and no physical or electronic attempts are made against I2P property or data centers.

Data Protection: We implemented multiple safeguards to minimize potential harm from data collection. All data were obtained exclusively from 15 floodfill routers that we deployed, which participated in normal netDB maintenance and tunnel construction without disrupting the I2P network. Data analysis was conducted entirely offline. We ensured strict confidentiality, with no data or metadata shared with third parties. All data were transmitted over SSH-encrypted channels and stored on a secure, access-controlled campus server. Our interaction with the data was purely observational: no modifications were made. All collected data were anonymized and deleted after paper submission.

References

[1] H. Wang, Z. Ling, X. Xu, Y. Pan, G. Liu, J. Luo, and X. Fu, "Time will tell: Large-scale de-anonymization of hidden i2p services via live behavior alignment (extended version)," https://doi.org/10.48550/arXiv.2512.15510, 2025.

[2] "I2p metrics," https://i2p-metrics.np-tokumei.net/, 2024, accessed: 2024/10/14.

[3] R. Dingledine, N. Mathewson, and P. F. Syverson, "Tor: The second-generation onion router," in *Proceedings of the 13th USENIX Security Symposium, USENIX Security*, 2004.

[4] Z. Ling, J. Luo, W. Yu, X. Fu, D. Xuan, and W. Jia, "A new cell counter based attack against tor," in *Proceedings of the 16th ACM SIGSAC Conference on Computer and Communications Security, CCS*, 2009.

[5] S. Murdoch and G. Danezis, "Low-cost traffic analysis of Tor," in *the 26th IEEE Symposium on Security and Privacy, S&P*, 2005.

[6] X. Wang, S. Chen, and S. Jajodia, "Network Flow Watermarking Attack on Low-Latency Anonymous Communication Systems," in *the 28th IEEE Symposium on Security and Privacy, S&P*, 2007.

[7] P. Mittal, A. Khurshid, J. Juen, M. Caesar, and N. Borisov, "Stealthy traffic analysis of low-latency anonymous communication using throughput fingerprinting," in *Proceedings of the 18th ACM SIGSAC Conference on Computer and Communications Security, CCS*, 2011.

[8] A. Biryukov, I. Pustogarov, and R.-P. Weinmann, "Trawling for tor hidden services: detection, measurement, deanonymization," in *the 34th IEEE Symposium on Security and Privacy, S&P*, 2013.

[9] A. Houmansadr and N. Borisov, "SWIRL: A Scalable Watermark to Detect Correlated Network Flows," in *Proceedings of the 2011 Annual Network and Distributed System Security Symposium, NDSS*, 2011.

[10] N. Borisov, G. Danezis, P. Mittal, and P. Tabriz, "Denial of service or denial of security?" in *Proceedings of the 14th ACM SIGSAC Conference on Computer and Communications Security, CCS*, 2007.

[11] S. Chakravarty, A. Stavrou, and A. D. Keromytis, "Traffic Analysis against Low-Latency Anonymity Networks Using Available Bandwidth Estimation," in *15th European Symposium on Research in Computer Security, ESORICS*, 2010.

[12] N. S. Evans, R. Dingledine, and C. Grothoff, "A Practical Congestion Attack on Tor Using Long Paths," in *Proceedings of the 18th USENIX Security Symposium, USENIX Security*, 2009.

[13] C. Wacek, H. Tan, K. Bauer, and M. Sherr, "An Empirical Evaluation of Relay Selection in Tor," in *Proceedings of the 20th Annual Network and Distributed System Security Symposium, NDSS*, 2013.

[14] M. Edman and P. Syverson, "As-awareness in Tor path selection," in *Proceedings of the 16th ACM Conference on Computer and Communications Security, CCS*, 2009.

[15] S. Matic, P. Kotzias, and J. Caballero, "CARONTE: detecting location leaks for deanonymizing tor hidden services," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, CCS*, 2015.

[16] A. Kwon, M. AlSabah, D. Lazar, M. Dacier, and S. Devadas, "Circuit Fingerprinting Attacks: Passive Deanonymization of Tor Hidden Services," in *Proceedings of the 24th USENIX Security Symposium, USENIX Security*, 2015.

[17] F. Rochet, R. Wails, A. Johnson, P. Mittal, and O. Pereira, "CLAPS: Client-Location-Aware Path Selection in Tor," in *Proceedings of the 27th ACM SIGSAC Conference on Computer and Communications Security, CCS*, 2020.

[18] Y. Sun, A. Edmundson, N. Feamster, M. Chiang, and P. Mittal, "Counter-RAPTOR: safeguarding tor against active routing attacks," in *the 38th IEEE Symposium on Security and Privacy, S&P*, 2017.

[19] S. E. Oh, T. Yang, N. Mathews, J. K. Holland, M. S. Rahman, N. Hopper, and M. Wright, "DeepCoFFEA: Improved Flow Correlation Attacks on Tor via Metric Learning and Amplification," in *the 43rd IEEE Symposium on Security and Privacy, S&P*, 2022.

[20] M. Nasr, A. Bahramali, and A. Houmansadr, "DeepCorr: Strong Flow Correlation Attacks on Tor Using Deep Learning," in *Proceedings of the 25th ACM SIGSAC Conference on Computer and Communications Security, CCS*, 2018.

[21] O. Berthold and H. Langos, "Dummy traffic against long term intersection attacks," in *the 3rd Workshop on Privacy Enhancing Technologie, PETS*, 2003.

[22] D. Lopes, J.-D. Dong, P. Medeiros, D. Castro, D. Barradas, B. Portela, J. Vinagre, B. Ferreira, N. Christin, and N. Santos, "Flow Correlation Attacks on Tor Onion Service Sessions with Sliding Subset Sum," in *Proceedings of the 31th Annual Network and Distributed System Security Symposium, NDSS*, 2024.

[23] L. Overlier and P. Syverson, "Locating hidden servers," in *2006 IEEE Symposium on Security and Privacy, S&P*, 2006.

[24] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker, "Low-resource routing attacks against tor," in *Proceedings of the 2007 ACM workshop on Privacy in electronic society, WPES*, 2007.

[25] R. Nithyanand, O. Starov, A. Zair, P. Gill, and M. Schapira, "Measuring and Mitigating AS-level Adversaries Against Tor," in *Proceedings of the 22nd Annual Network and Distributed System Security Symposium, NDSS*, 2015.

[26] M. Backes, A. Kate, S. Meiser, and E. Mohammadi, "(Nothing else) MATor(s): Monitoring the Anonymity of Tor's Path Selection," in *Proceedings of the 21st ACM SIGSAC Conference on Computer and Communications Security, CCS*, 2014.

[27] Y. Sun, A. Edmundson, L. Vanbever, O. Li, J. Rexford, M. Chiang, and P. Mittal, "{RAPTOR}: Routing Attacks on Privacy in Tor," in *Proceedings of the 24th USENIX Security Symposium, USENIX Security*, 2015.

[28] V. Shmatikov and M.-H. Wang, "Timing analysis in low-latency mix networks: Attacks and defenses," in *the 15th European Symposium on Research in Computer Security, ESORICS*, 2006.

[29] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson, "Users get routed: Traffic correlation on Tor by realistic adversaries," in *Proceedings of the 20th ACM SIGSAC Conference on Computer and Communications Security, CCS*, 2013.

[30] F. Rochet and O. Pereira, "Waterfilling: balancing the tor network with maximum diversity," in *the 17th Privacy Enhancing Technologies Symposium, PETS*, 2017.

[31] Y. Zhang and V. Paxson, "Detecting Stepping Stones," in *Proceedings of the 9th USENIX Security Symposium, USENIX Security*, 2000.

[32] A. Blum, D. Song, and S. Venkataraman, "Detection of Interactive Stepping Stones: Algorithms and Confidence Bounds," in *the 7th International Symposium on Research in Attacks, Intrusions, and Defenses, RAID*, 2004.

[33] N. Mathewson and R. Dingledine, "Practical traffic analysis: Extending and resisting statistical disclosure," in *the 4th Workshop on Privacy Enhancing Technologies, PETS*, 2004.

[34] G. Danezis, C. Diaz, and C. Troncoso, "Two-Sided Statistical Disclosure Attack," in *the 7th Workshop on Privacy Enhancing Technologies, PETS*, 2007.

[35] S. J. Murdoch and P. Zieliński, "Sampled Traffic Analysis by Internet-Exchange-Level Adversaries," in *the 7th Workshop on Privacy Enhancing Technologies, PETS*, 2007.

[36] R. Jansen, T. Vaidya, and M. Sherr, "Point break: A study of bandwidth {Denial-of-Service} attacks against tor," in *Proceedings of the 28th USENIX Security Symposium, USENIX Security*, 2019.

[37] R. Jansen, F. Tschorsch, A. Johnson, and B. Scheuermann, "The sniper attack: Anonymously deanonymizing and disabling the tor network," in *Proceedings of the 2014 Annual Network and Distributed System Security Symposium, NDSS*, 2014.

[38] M. V. Barbera, V. P. Kemerlis, V. Pappas, and A. D. Keromytis, "Cellflood: Attacking tor onion routers on the cheap," in *the 18th European Symposium on Research in Computer Security, ESORICS*, 2013.

[39] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, "Measurement, modeling, and analysis of a peer-to-peer file-sharing workload," in *Proceedings of the nineteenth ACM symposium on Operating systems principles, SOSP*, 2003.

[40] S. Rhea, D. Geels, T. Roscoe, J. Kubiatowicz *et al.*, "Handling churn in a DHT," in *2025 USENIX Annual Technical Conference, USENIX ATC*, 2003.

[41] S. Saroiu, K. P. Gummadi, and S. D. Gribble, "Measuring and analyzing the characteristics of napster and gnutella hosts," *Multimedia Systems*, vol. 9, no. 2, pp. 170–184, 2003.

[42] S. Sen and J. Wang, "Analyzing peer-to-peer traffic across large networks," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment, IMW*, 2002.

[43] K. Sripanidkulchai, B. Maggs, and H. Zhang, "An analysis of live streaming workloads on the internet," in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement, IMC*, 2004.

[44] I2PRelease, "I2P 2.8.0 Released," 2025, accessed: 2025-07-11. [Online]. Available: https://geti2p.net/en/blog/post/2025/02/03/i2p-release-2.8.0

[45] I2P, "The Invisible Internet Project," 2025, accessed: 2025-08-06. [Online]. Available: https://geti2p.net/en/

[46] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *the 1st International Workshop on Peer-to-Peer Systems, IPTPS*, 2002.

[47] N. P. Hoang, P. Kintis, M. Antonakakis, and M. Polychronakis, "An empirical study of the I2P anonymity network and its censorship resistance," in *Proceedings of the 18th Internet Measurement Conference, IMC*, 2018.

[48] B. Zantout and R. Haraty, "I2P data communication system," in *The Tenth International Conference on Networks, ICN*, 2011.

[49] M. Herrmann and C. Grothoff, "Privacy-implications of performance-based peer selection by onion-routers: a real-world case study using i2p," in *the 11th Privacy Enhancing Technologies Symposium, PETS*, 2011.

[50] C. Egger, J. Schlumberger, C. Kruegel, and G. Vigna, "Practical attacks against the I2P network," in *Proceedings of the 16th International Symposium on Research in Attacks, Intrusions and Defenses, RAID*, 2013.

[51] A. Crenshaw, "Darknets and hidden servers: Identifying the true IP/network identity of I2P service hosts," *Black Hat DC 2011*, vol. 201, no. 1, pp. 1–23, 2011.

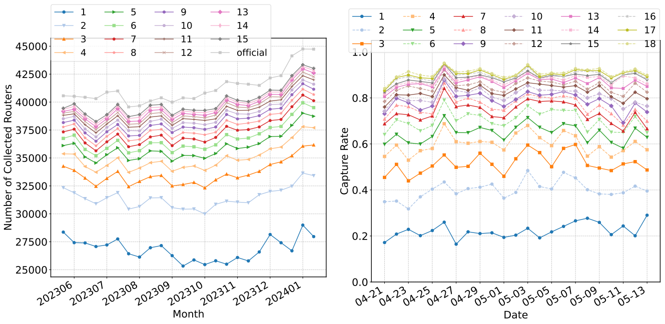
[52] M. Simioni, P. Gladyshev, B. Habibnia, and P. R. N. de Souza, "Monitoring an anonymity network: Toward the deanonymization of hidden services," *Forensic Science International: Digital Investigation*, vol. 38, p. 301135, 2021.

[53] S. J. Murdoch, "Hot or not: Revealing hidden services by their clock skew," in *Proceedings of the 13th ACM SIGSAC Conference on Computer and Communications Security, CCS*, 2006.

[54] I2P Project, "Vulnerability Response Process," 2025, accessed: 2025-08-06. [Online]. Available: https://geti2p.net/en/research/vrp

Appendix A
Low-cost deployment strategy for floodfill routers

We evaluate the data collection effectiveness of 15 floodfill routers using two key metrics: the network-wide router coverage and the *RouterInfo* capture rate from controlled routers. Over an 8-month monitoring period (2023.06–2024.02), we continuously recorded the number of unique I2P routers detected each day and compared it with the official statistics [2] to validate our network coverage, as shown in Fig. 11a. With 15 floodfill routers, our system achieved 98% coverage of the officially reported router count, with only marginal gains observed beyond 13 routers. We further assessed the capture rate of *RouterInfo* data by comparing the records collected by our floodfill routers with the actual number of *RouterInfo* publications logged locally by the controlled host routers. Fig. 11b presents the average capture rates achieved using different numbers of floodfill routers. A 15-router deployment yields a 90% capture rate, comparable to that of 18 routers. Therefore, we use 15 floodfill routers in all real-world experiments to ensure reliable data collection with minimal resource overhead.



(a)

(b)

Fig. 11: Floodfill routers' effectiveness. (a) Comparison between our collected data and official data. (b) RouterInfo Capture Rate vs. Number of Controlled Routers.

Appendix B
Probability that no two processes are the same after m cycles

A. Problem Definition

Given n on-off stochastic processes. Their *on* periods follow the same discrete probability distribution and have k values with probabilities $a_1,\dots,a_k$. Their off periods follow the same discrete probability distribution and have l values with probabilities $b_1,\dots,b_l$. The n processes are independent and their on-off cycles are independent. One cycle contains one *on* period and one off period.

What is the probability $P_{\mathcal{A}}$ that one particular process A does not produce the same length-m sequence with any of the other n-1 processes?

B. Solution

Step 1. Model a single cycle. One cycle C is an ordered pair (O,F) where O is an on-period chosen from $\{1,\dots,k\}$ with probabilities $a_1,\dots,a_k$, and F is an off-period chosen from $\{1,\dots,l\}$ with probabilities $b_1,\dots,b_l$. Since O and F are independent,

$$P(C=(i,j)) = a_i b_j, \tag{10}$$

where $i=1,\dots,k$ and $j=1,\dots,l$. Note:

$$\sum_{i=1}^{k} a_i = 1, \qquad \sum_{j=1}^{l} b_j = 1 \tag{11}$$

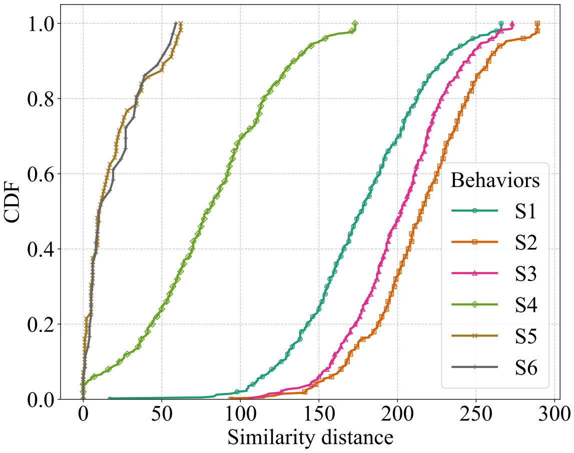


Fig. 12: Distribution of similarity distances between inferred live behaviors of router pairs sharing the same behavior

Step 2. Probability P_C two independent processes match on one cycle. Let process A produce cycle C and an independent process B produce cycle C'. The probability they coincide on a particular cycle value (i,j) is

$$P(C=(i,j))\,P(C'=(i,j)) = (a_i b_j)^2. \tag{12}$$

Summing over all possible outcomes gives

$$P_C = \sum_{i=1}^{k}\sum_{j=1}^{l}(a_i b_j)^2 = \left(\sum_{i=1}^{k} a_i^2\right)\left(\sum_{j=1}^{l} b_j^2\right). \tag{13}$$

Define

$$S_a = \sum_{i=1}^{k} a_i^2, \qquad S_b = \sum_{j=1}^{l} b_j^2. \tag{14}$$

Therefore,

$$P_C = S_a S_b. \tag{15}$$

Step 3. Probability P_m that two processes of length-m match. A length-m sequence consists of m independent cycles. Two processes produce the same sequence exactly when they match in all m cycles. Since cycles are independent,

$$P_m = (S_a S_b)^m. \tag{16}$$

Step 4. Probability $P_{\mathcal{A}}$ that none of the other n-1 processes match process A. Fix the realized length-m sequence of process A. For any other process B, the probability it matches A on that fixed sequence is $P_m = (S_a S_b)^m$. Thus the probability it does *not* match is $1-P_m$. Since all $n-1$ other processes are independent,

$$P_{\mathcal{A}} = \left(1-P_m\right)^{n-1}. \tag{17}$$

Appendix C
Discussion on the similarity threshold for distinguishing router behaviors

To assess whether two I2P routers exhibit indistinguishable behavioral patterns, we analyze the similarity distance between the inferred live behaviors of controlled routers that share the same ground-truth behavior. For each day in the evaluation dataset(refer to §V-A), we compute the pairwise similarity distances among all controlled routers with identical ground-truth behaviors, as shown in Fig. 12. Routers with stable online patterns (**S5** and **S6**) display consistently low distances, with 85% below 37 and a maximum of 62. In contrast, routers exhibiting frequent on–off transitions (**S1**–**S4**) show substantially larger distances ranging from 6 to 389. The results indicate that even when routers share the same ground-truth behavior, the similarity distance between their inferred live behaviors increases with the number of on–off switches. We therefore examined the distribution of online sessions in the real I2P network. Except for routers that remain continuously online, 99.74% of routers experience fewer than four online sessions per day. Motivated by this observation, we recomputed similarity distances for controlled routers sharing the same behavior and exhibiting one to four sessions. Nearly all distances fall below 166, with only rare outliers reaching 189. Based on these results, we set the similarity threshold to 200, which provides a conservative upper bound ensuring that routers following identical behaviors are unlikely to be incorrectly excluded from each other's anonymity sets, thereby minimizing false negatives. Although such a relaxed threshold may allow inclusion of some dissimilar routers, behavioral divergence that accumulates over longer observation windows can still enable an adversary to gradually shrink the target's anonymity set.

Appendix D
Live behavior cases with data loss and non-data-loss cases

Table VI provides a comprehensive enumeration of all live-behavior cases that challenge our inference framework. Specifically, it includes (i) cases where the coarse-grained session inference becomes inaccurate under no–data-loss conditions (Cases 1–5 and Case 13), and (ii) cases where the fine-grained inference method may misidentify join or leave events due to data loss (Cases 6–12 and Cases 14–16). Whereas the main text (Section IV-C and Section IV-D) discusses these cases individually when explaining the design rationale of each live behavior inference component, this appendix consolidates them in one place to support systematic cross-case comparison.

Each row of Table VI represents a distinct live behavior pattern and the resulting *RouterInfo* publication pattern of that behavior in either Java- or C++-based routers, along with the root cause of the inference error, a schematic comparison of inferred versus actual live behaviors, and a concise description of how the inference error arises. The last 3 columns indicate which technical components of our framework (join behavior identification, leave behavior identification, and online session complement) can correctly handle the case. The table thus highlights the functional contribution of every technical component and clarifies why all components are necessary to achieve robust live behavior inference.

TABLE VI: The situations needs to be handled and the functional contributions of each technical component across varying situations.

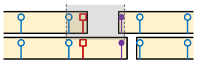
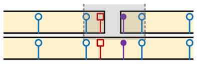
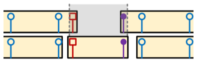
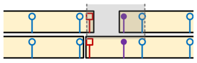
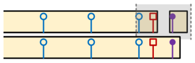
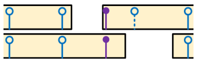
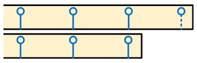
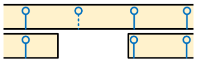
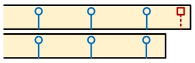
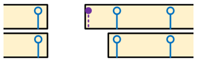
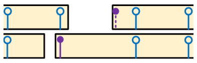
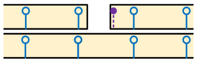
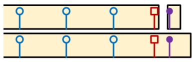
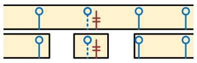
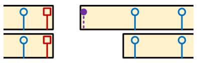
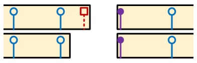
Index	Lang.	Data Loss	Root Cause	Schematic Diagram	Description	Coarse-Grained	+ Join Behavior	+ Leave Behavior	+ Session Complement
1	Java	✗	**Routine *RouterInfo* mismatch**		$I(R_r^i, R_f^{i+1}) = I_r$	✗	✓	-	-
2					$I(R_r^i, R_r^{i+1}) = I_r$	✗	✓	-	-
3			**Routine *RouterInfo* misidentification**		$I(R_\ell^i, R_f^{i+1}) = I_r$	✗	✓	-	-
4					$I(R_\ell^i, R_r^{i+1}) = I_r$	✗	✓	-	-
5			**No routine feature**		No routine *RouterInfo* is published within the $(i+1)^{\text{th}}$ online session. $I(R_r^i, R_f^{i+1}) < I_r$	✗	✗	✓	-
6		✓	**Loss of a routine *RouterInfo***		The first R_r^{i+1} is lost. (non-floodfill only).	✗	✗	✗	✓
7					The last R_r^i is lost. (non-floodfill only)	✗	✗	✗	✗
8					Loss of any R_r^i other than the first or the last.	✗	✗	✗	✓
9			**Loss of the final *RouterInfo***		The R_f^i is lost. Several R_r^i may also lost. (floodfill only).	✗	✗	✗	✓
10			**Loss of the initial *RouterInfo***		R_f^{i+1} is lost, and $I(R_r^i, R_r^{i+1}) \neq I_r$.	✗	✗	✗	✓
11					R_f^{i+1} is lost, be mistakenly handled as Case 6 (non-floodfill only).	✗	✗	✗	✗
12					R_f^{i+1} is lost, and $I(R_r^i, R_r^{i+1}) = I_r$, (non-floodfill only).	✗	✗	✗	✗
13	C++	✗	**No routine feature**		Identical to case 5.	✗	✗	✓	-
14		✓	**Loss of routine *RouterInfo***		A R_r^i is lost, the next *RouterInfo* has different congestion flag with the previous R_r^i.	✗	✗	✗	✓
15			**Loss of the initial *RouterInfo***		R_f^{i+1} is lost.	✗	✗	✗	✓
16			**Loss of the final *RouterInfo***		R_ℓ^{i+1} is lost.	✗	✗	✗	✓

Diagram explanation: Each diagram displays both the actual live behavior of the router (top section) and the inferred live behavior (bottom section). Each yellow rectangles represent router's one online session. Within each online session, the blue short line ending with a circle indicates the publish time of routine *RouterInfo* data, the red short line with a hollow square marks the publish time of the final *RouterInfo* data, and the purple short line terminating in a solid circle denotes the publish time of initial *RouterInfo* data within the session. Dashed line represents the *RouterInfo* data that is lost. Additionally, in cases 1-5, gray rectangles between two online sessions indicate approximately one routine interval. Specifically, the time interval between the two *RouterInfo* data located at the leftmost and rightmost edges of the gray rectangle equals one routine interval.

Formal definition: We define the function $I(\textit{RouterInfo 1}, \textit{RouterInfo 2})$ to calculate the time interval between two *RouterInfo* data, with I_r representing the routine interval. Next, we assign identifiers to important *RouterInfo* data. For a given *RouterInfo* R, the superscript (e.g., i) indicates *RouterInfo* data published within the i^{th} online session. Subscripts (e.g., r, f and ℓ) to indicate whether the *RouterInfo* is a routine, the initial or the last *RouterInfo* data within an online session, respectively.