2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS) | 978-1-6654-7177-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICDCS54860.2022.00064

RAPTEE: Leveraging trusted execution environments for Byzantine-tolerant peer sampling services

Matthieu Pigaglio*, Joachim Bruneau-Queyreix†, Yérom-David Bromberg‡,
Davide Frey‡, Etienne Rivière*, Laurent Réveillère†
* ICTEAM, UCLouvain, Belgium
† Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR5800, F-33400 Talence, France
‡ Univ. Rennes, CNRS, Inria, IRISA, Rennes, France

***Abstract*—Peer sampling is a first-class abstraction used in distributed systems for overlay management and information dissemination. The goal of peer sampling is to continuously build and refresh a partial and local view of the full membership of a dynamic, large-scale distributed system. Malicious nodes under the control of an adversary may aim at being over-represented in the views of correct nodes, increasing their impact on the proper operation of protocols built over peer sampling. State-of-the-art Byzantine resilient peer sampling protocols reduce this bias as long as Byzantines are not overly present. This paper studies the benefits brought to the resilience of peer sampling services when considering that a small portion of trusted nodes can run code whose authenticity and integrity can be assessed within a trusted execution environment, and specifically Intel's software guard extensions technology (SGX). We present RAPTEE, a protocol that builds and leverages trusted gossip-based communications to hamper an adversary's ability to increase its system-wide representation in the views of all nodes. We apply RAPTEE to BRAHMS, the most resilient peer sampling protocol to date. Experiments with 10,000 nodes show that with only 1% of SGX-capable devices, RAPTEE can reduce the proportion of identifiers of Byzantine nodes in the view of honest ones by up to 17%, when the system contains 10% of Byzantine nodes. In addition, the security guarantees of RAPTEE hold even in the presence of a powerful attacker attempting to identify trusted nodes and injecting view-poisoned trusted nodes.**

I. INTRO

Peer sampling is a first-class abstraction for the construction of large-scale distributed systems. It is notably used for overlay management [14], [22] and information dissemination [11], [17]. Typically, nodes possess partial knowledge (also referred to as their *view*) of the global and dynamic system membership. The goal of the peer-sampling distributed service is to bootstrap and continuously refresh this local view so that it corresponds as much as possible to a *uniform* sample of alive nodes in the system. The implementation of the peer-sampling service is typically based on gossip protocols leveraging simple, periodic peer-wise exchanges of information. A plethora of protocols have been published and studied [3], [15], [21] under considerations of crash faults, churns, performance, ergodicity, and desirable structural properties such as balanced in-degree, low diameter, and ability to quickly remove departed nodes from the views of alive ones.

The resilience of peer-sampling protocols to Byzantine faults is crucial for the security of applications relying upon it. Indeed, malicious nodes that succeed in becoming over-represented in the views of honest nodes can gain control of the upper-layer protocols. For example, the peer-sampling protocol of Bitcoin was discovered to be exposed to eclipse attacks [12], opening the door to multiple types of selfish mining and double-spending attacks at the consensus level. In state-of-the-art Byzantine-resilient peer-sampling protocols [3], [6], the views of honest nodes can swiftly be poisoned with Byzantine identifiers as the proportion of malicious nodes in the system grows. With BRAHMS [6], the most Byzantine-resilient protocol, the views of honest nodes may contain up to 81% of Byzantine identifiers (*IDs*) when 18% of the nodes in the system are malicious.

The advent of trusted execution environments (TEEs) has exposed many desirable properties for distributed systems, notably remote attestation, authentication, integrity, and confidentiality of the execution environment. This paper shows how the resilience of peer-sampling protocols to Byzantine behaviors can be improved with these TEE properties. In this paper, we consider Intel's Software Guard Extensions (SGX) [9] released in 2015, but any TEE technology providing similar code integrity validation properties would be a fit for our approach, e.g., ARM's TrustZone [18].

We present RAPTEE, a novel protocol that integrates trusted communications in the peer-sampling-service operation and interoperates them with BRAHMS. We consider a system composed of honest and Byzantine nodes, along with a small proportion of trusted nodes (*i.e.*, SGX-capable devices). Because trusted nodes cannot act maliciously, RAPTEE accelerates the dissemination of knowledge they possess while slowing down the action of Byzantine nodes and preventing the identification of trusted nodes by an adversary.

In RAPTEE, all nodes execute a modified version of BRAHMS. However, trusted nodes are not bound to strictly execute BRAHMS when interacting with other trusted nodes, even though they can possess Byzantine IDs in their views. Contrarily, when communicating with untrusted nodes, additional Byzantine-tolerance countermeasures are required. They consist of adaptively ignoring part of the IDs sent by untrusted nodes. This voluntary loss of information enables higher resilience. Indeed, it reduces the ability of an attacker to pollute the views of trusted nodes by sending them bulks of

2575-8411/22/$31.00 ©2022 IEEE
DOI 10.1109/ICDCS54860.2022.00064

Authorized licensed use limited to: Universite Picardie Jules Verne. Downloaded on November 15,2022 at 15:35:13 UTC from IEEE Xplore. Restrictions apply.

malicious IDs, which makes trusted nodes act as sources of less biased identifiers than untrusted nodes.

Because trusted nodes can improve the system's Byzantine resilience, they are of utmost interest for malicious nodes in performing targeted attacks and eclipsing them from the system. It becomes clear that they cannot advertise themselves as trustworthy actors and should remain hidden in the mass. Accordingly, RAPTEE leverages the ability of trusted nodes to learn their mutual trusted capacity without revealing it to others. Ignoring IDs also hampers the dissemination of correct IDs transmitted by honest nodes, which could have a significant impact on system convergence in the time it takes to discover a majority of nodes. To address this problem, RAPTEE makes trusted nodes exchange information in a dissemination-efficient way using gossip-based peer-sampling interactions following the framework of Jelasity *et al.* [15].

Our experiments with 10,000 nodes on the Grid 5000 [5] testbed show that with no more than 1% of SGX-capable devices, RAPTEE can reduce the proportion of Byzantine IDs in the views of honest nodes by up to 17% when the ratio of Byzantine nodes in the system is 10%. This resilience gain comes at the expense of a limited overhead of 10% in terms of the required number of rounds to view stability and 12% for system discovery. We assess the security risks of RAPTEE against two types of attack: trusted-node identification and view-poisoned trusted node injection. We show that the identification attack is particularly inefficient before the system converges and is impossible once this point is reached. Considering injecting trusted nodes with polluted views, we show that it has little to no impact on the system's resilience and that trusted nodes can self-heal with RAPTEE.

Outline. First, we provide some requisite background knowledge on BRAHMS and on the general gossip-based peer-sampling framework in Section II. Then, we provide a high-level overview of RAPTEE with its models and objectives in Section III. We describe RAPTEE and detail how it complements BRAHMS with trusted communications in Section IV. Section V exposes the experimental methodology used to evaluate the performance and resilience of RAPTEE under different configurations and discusses the results. We provide a security assessment of RAPTEE to trusted-node-identification and corrupted-trusted-node-injection attacks in Section VI. We summarize our findings in Section VII and review the state of the art in Section VIII before concluding in Section IX.

II. BACKGROUND

We start by presenting background on gossip-based peer sampling and on the BRAHMS protocol [6].

Gossip-based peer sampling: Gossip is a mode of operation for large-scale distributed systems based on repeated, periodic peer-to-peer interactions between pairs of peers [19]. Despite the general simplicity of these local interactions, Gossip protocols exhibit very good global robustness properties (*i.e.*, the ability to resist high levels of churn or survive network partitions) thanks to their self-organizing nature [4], [14]. These properties often depend, however, on the quality of the random process that determines the participants in these peer-wise interactions. This selection of communication partners at each period should be as close as possible to a uniform random draw amongst all participating nodes, including newly joined ones and excluding departed or failed ones. The *peer-sampling service* plays a fundamental role in ensuring this property.

Interestingly, the most efficient way to implement a peer-sampling service for use by other protocols, gossip-based or not, is to design it as a gossip-based protocol itself. The goal of such a protocol is to equip each node of the system with a view, *e.g.*, a sample of other nodes in the system. The graph formed by the who-knows-whom relations with these views should be as close as possible to a random graph of fixed out degree. As such, it should have balanced in-degrees (to prevent nodes from being under or over-represented in the views of other nodes), a low diameter, and a low clustering coefficient. In addition, these properties must be enforced while quickly including newly joined peers in the views of other nodes (so as to match their in-degree, and therefore the probability of being selected as a gossip partner, to that of existing nodes) and quickly removing departed or failed nodes. Addressing these different requirements simultaneously is, unfortunately, impossible and gossip-based peer-sampling protocols are the result of a compromise between them.

Jelasity *et al.* [14] proposed a universal framework for the design of gossip-based protocols that encompasses previous designs with emphasis on a specific property (*e.g.*, balanced in degree and low clustering for Cyclon [21] or efficient dynamic membership for Newscast [20]). This framework defines two parameters, H (for Heal) and S (for Shuffling), that drive the exchange of (partial) views of size c between peers, in addition to other parameters such as the policy for partner selection or the policy for the link deprecation in individual nodes' views. On the basis of the recommendations of Jelasity *et al.* [14], we apply this framework with the following criteria. (1) Peers initiate an exchange with the peer that has been for the longest time in their view, on the basis of an age parameter associated with its entries. This effectively enables a round robin *probing* of view neighbors. (2) They exchange half of their view, and the initiator peer inserts a link to itself in the view sent to the gossip partner. (3) The exchange favors the *shuffling* of the links, *i.e.*, a link sent from the initiator will be kept only by the partner, and *vice-versa*.

BRAHMS: This protocol presented by Bortnikov *et al.* [6] is designed for large and dynamic systems prone to Byzantine failures and sybil attacks. With high probability, BRAHMS prevents an attacker from creating a partition between correct nodes, and allows each node's view to converge to a uniform random draw of all participating nodes over time. The main ideas behind Brahms are to use push-pull gossip-based membership with some additional defense mechanisms to prevent local views from being exclusively composed of Byzantine IDs, and to correct bias in the views resulting from attacks. We briefly describe the two components of BRAHMS employed by each node before detailing these defense mechanisms.

BRAHMS has two components. The first is a *gossip* compo-

Authorized licensed use limited to: Universite Picardie Jules Verne. Downloaded on November 15,2022 at 15:35:13 UTC from IEEE Xplore. Restrictions apply.

nent. Each node spreads locally known IDs across the system and maintains a dynamic view V of l_1 entries. The second is a *local sampling* component that enables each node to maintain a sample list S of l_2 entries uniformly sampled from the received IDs. Uniformity is obtained by a min-wise independent permutation [8] technique.

Initially, each node possesses a list containing node IDs and addresses obtained from a bootstrap node. Periodically, a BRAHMS node, through the gossip component, selects from its dynamic view V both $\alpha \times l_1$ nodes, to send them push messages containing its own ID, and $\beta \times l_1$ nodes, to send them pull-request messages in order to retrieve their views. At the end of each round, the sample list S is updated via the sampling component, while the dynamic view V is renewed by randomly selecting: (i) $\alpha \times l_1$ IDs received from push messages, (ii) $\beta \times l_1$ IDs from pull answers, and (iii) $\gamma \times l_1$ IDs from the sample list (referred to as the history sample), with $\alpha + \beta + \gamma = 1$, as shown in Figure 1.

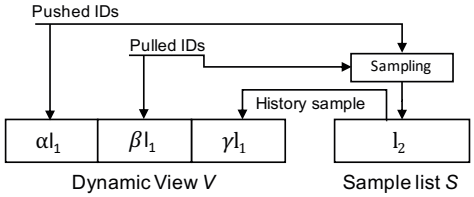


Fig. 1: BRAHMS view computation

As depicted in Figure 2, the sampling component consists of l_2 samplers. It takes as input a stream of identifiers received from push or pull messages and produces a sample list of size l_2. At initialization, each sampler chooses a hash function at random. A sampler takes an identifier as input and computes its hash. If the calculated hash is smaller than the one corresponding to a previously stored identifier, the sampler produces the new identifier and stores it in its local memory. Otherwise, the sampler outputs the stored ID.

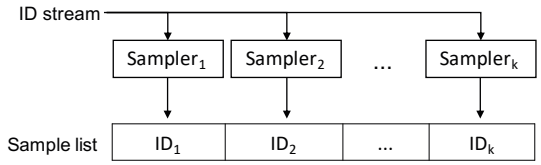


Fig. 2: BRAHMS sampling component

The four defense mechanisms that prevent partitioning and allow convergence to uniform sampling are the following: (i) limited pushes, (ii) attack detection and blocking, (iii) controlling the contribution of pulls versus pushes, and (iv) history sampling. We detail these mechanisms in the following. (i) An adversary could forge identities or flood the system with push requests, leaving correct IDs propagated mainly through pulls and diminishing their representation exponentially. BRAHMS assumes a mechanism that limits the message sending rate of

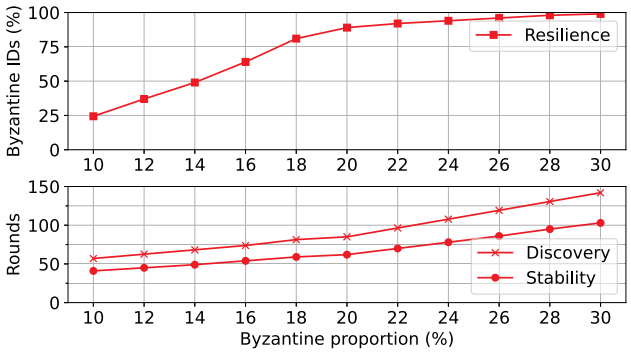


Fig. 3: BRAHMS resilience, time to discovery and time to stability under Byzantine faults

nodes, for example, via computational challenges like Merkle's puzzles, virtual currency, *etc*. (ii) Limiting push messages prevents a simultaneous attack on all correct nodes but does not protect against flooding a targeted node. To do so, BRAHMS blocks dynamic view updates if more than the expected $\alpha \times l_1$ pushes are received. This policy slows down progress but its expected impact in the absence of attacks is bounded, and thanks to limited pushes, some nodes make progress even under attack. (iii) Node views are threatened by pulls from neighbors more than by adversarial pushes. Pushes from correct nodes are correct, but pull answers from correct nodes may contain some identifiers of Byzantine nodes. Hence, the contribution of pushes and pulls to V must be balanced. Brahms updates V with randomly chosen $\alpha \times l_1$ pushed IDs to protect targeted nodes, and $\beta \times l_1$ pulled IDs to protect the rest. (iv) The attack detection and blocking technique slows down a targeted attack but cannot prevent it completely. A node victim of targeted pushes will pull more IDs from Byzantine nodes, send fewer pushes to correct ones, causing its system-wide representation to decrease, hence receiving fewer correct pushes. Brahms overcomes such an attack using a self-healing mechanism by incorporating in the view an unbiased historical sample of $\gamma \times l_1$ IDs from the sample list S. Once some correct ID becomes the permanent sample of the node under attack (or if the node's ID becomes a permanent sample of another correct node) the threat of isolation is eliminated.

Results on Figure 3 show resilience (percentage of Byzantine IDs in the views of correct nodes), time to discovery (number of rounds required for all nodes to discover at least 75% of non-Byzantine IDs) and time to view stability (number of rounds necessary for all non-Byzantine node views to be polluted within 10% of the average proportion of Byzantine IDs in the views non-Byzantine nodes) of BRAHMS under different shares of Byzantine nodes in the system. The Brahms parameters are set as recommended in the original paper [6], namely $\alpha = \beta = 0.4$ and $\gamma = 0.2$.

III. MODELS AND OBJECTIVES

We present our system model and objectives, followed by our trust and adversarial models.

Authorized licensed use limited to: Universite Picardie Jules Verne. Downloaded on November 15,2022 at 15:35:13 UTC from IEEE Xplore. Restrictions apply.

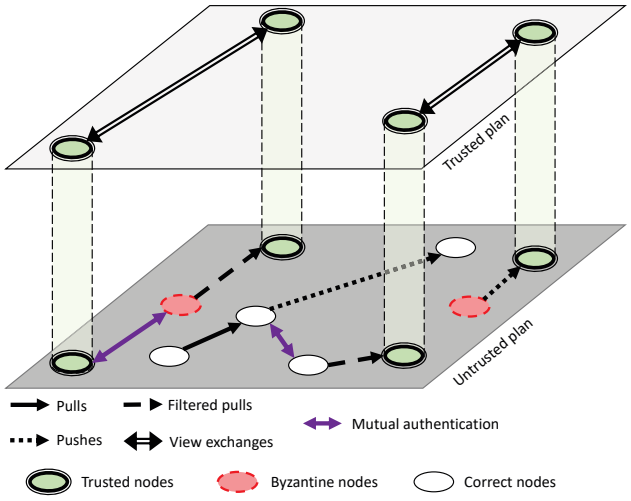


Fig. 4: Overview of the RAPTEE protocol

A. System, operating models and RAPTEE's objectives

We consider a system of N nodes composed of a fraction f of Byzantine nodes that can deviate from the protocol in any possible way, a fraction t of trusted nodes, and a fraction $h = 1 - f - t$ of honest (or correct) nodes that execute BRAHMS. Each node is identified by a unique ID, chosen when the node becomes active for the first time. Figures 4 gives an overview of RAPTEE and depicts the interactions between the different types of nodes.

Honest nodes aim to get a uniform sample of the global membership. They execute a modified version of BRAHMS to exchange identifiers. RAPTEE relies on both the push-pull gossip and sampling components of BRAHMS as detailed in Section II, as well as on its defense mechanisms. Push messages include only the sender's identifier, while responses to pull requests contain the full view of the requested node.

Interactions between different types of nodes are designed to slow down the discovery of IDs possessed by untrusted nodes and accelerate the dissemination of IDs possessed by trusted nodes. To do so, trusted nodes filter out pull answers from untrusted ones by evicting part of the received IDs. During trusted communications, trusted nodes exchange view parts following the instantiation of the Gossip-based Peer Sampling framework [15] and as detailed in the previous section. To decide on the types of node interactions (*e.g.*, trusted to untrusted, trusted to trusted, *etc.*), RAPTEE includes a mutual authentication protocol that precedes all pull requests.

B. Trust and adversarial models

We consider an adversary controlling all Byzantine nodes with two objectives: successfully over-representing the system-wide fraction of Byzantine IDs in the partial membership knowledge of all nodes and identifying trusted nodes to evict them from the system. The adversary has access to the system's global membership, including Byzantine and correct nodes, but is unaware of the location or amount of SGX-capable devices. In the original BRAHMS paper [6], the authors prove that a balanced attack, which spreads faulty pushes evenly among correct nodes, maximizes the expected system-wide fraction of faulty IDs. Additionally, thanks to its history sample mechanism, BRAHMS sustains targeted attacks where the adversary tries to partition the network by targeting a subset of nodes and sending more pushes than in a balanced attack. Because RAPTEE is built on top of BRAHMS and inherits its properties, our adversary exclusively advertises Byzantine IDs to honest ones via answers to pull requests and evenly balanced push messages. We identify two additional attack vectors stemming from the addition of trusted nodes into the system. The first one consists of identifying trusted nodes to isolate them from non-Byzantine nodes, launching targeted attacks on them to pollute their views, and inserting them back into the system. The second one exploits the possibility of purchasing SGX-capable devices and bootstrapping their lifecycle by surrounding them with Byzantine nodes to almost fill their views with Byzantine IDs.

We rule out Sybil attacks [10] by relying on the Sybil resilience of BRAHMS that limits the message sending rate of nodes via computational challenges like Merkle's puzzles, virtual currency, *etc*. We assume that trusted nodes can only crash fault. They cannot act maliciously even though Byzantine IDs can bias their view. Our implementation uses specifically Intel SGX [9]. We trust Intel for the certification of genuine SGX-enabled CPUs, and we assume that the code running inside enclaves is properly attested before being provided with secrets. We assume Byzantine nodes can neither break cryptographic primitives nor read data available in the trusted environment of SGX-capable devices. We also assume that it is impractical for an adversary to extract secrets from the enclave using side-channel attacks, having no physical access or means to colocate a process on a target machine. Communications between any two nodes, including trusted ones, are ciphered with symmetric encryption to protect against an eavesdropping adversary. We consider that an adversary does not have global access to communication links in the network that do not involve itself and cannot, therefore, infer information from communication patterns.

IV. INTEROPERATING TRUSTED COMMUNICATIONS WITH BYZANTINE RESILIENT PEER SAMPLING

We detail in this section the main components of RAPTEE: mutual authentication, which allows trusted nodes to be recognized in the mass; trusted communication, which accelerates the dissemination of knowledge between trusted nodes; and Byzantine eviction, which slows down view poisoning.

A. Mutual authentication

To enable trusted nodes to know whether they are communicating with other trusted nodes, we design a secure mutual authentication protocol that is executed by all nodes before issuing a pull request to a selected neighbor in the dynamic node view. We assume that all nodes have one symmetric secret key. Each untrusted node generates a random secret key during the initialization phase. Conversely, trusted nodes

Authorized licensed use limited to: Universite Picardie Jules Verne. Downloaded on November 15,2022 at 15:35:13 UTC from IEEE Xplore. Restrictions apply.

share a common secret key that is provisioned during the remote-attestation phase. The mutual authentication protocol between two nodes A and B operates as follows. First, A generates a pseudo-random number r_A and sends it to B via a cryptographic challenge. In turn, B generates another pseudo-random number r_B and computes the hash of the concatenation of r_A and r_B, $H(r_A \cdot r_B)$, and encrypts it with its own secret key obtaining $[H(r_A \cdot r_B)]_{K_B}$. Then, B sends r_B and $[H(r_A \cdot r_B)]_{K_B}$ to A. Upon reception, A computes $H(r_A \cdot r_B)$ and deciphers $[H(r_A \cdot r_B)]_{K_B}$ using its own secret key K_A. If the two values are identical (*i.e.*, A and B share the same secret key), A can identify B as trusted. Then, A sends $[H(r_B \cdot r_A)]_{K_A}$ to B. Like A, B deciphers this encrypted hash using its own secret key K_B and compares it with $H(r_B \cdot r_A)$. If the two are equal, B can also identify A as trusted.

B. Trusted communications

We aim to accelerate the dissemination of identifiers among trusted nodes, and spread these to untrusted nodes. When two trusted nodes authenticate each other in a round, they can trust each other to not deviate from the protocol other than through crash failures. For this reason, trusted nodes perform a trusted peer-sampling phase during the current round, in which they exchange and sample identifiers in a manner different from that described for BRAHMS. During the trusted communication phase, each node follows the two following measures. First, it swaps half of its view with half of the remote node's view. Then, it transmits the received IDs to the list of pulled IDs in the BRAHMS protocol. The first measure allows trusted nodes to gossip more of the IDs they receive from other trusted nodes during the round (via pull messages). The second takes into account the IDs transmitted by other trusted nodes to update the sample list, and renews the dynamic view during the random selection of $\beta \times l_1$ pulled IDs.

C. Byzantine eviction

Since pull answers from Byzantine nodes contain exclusively Byzantine IDs, one could think that if trusted nodes did not send any pull requests, they could prevent the poisoning of their views. However, this simple approach would open the door for an attacker to identify trusted nodes and eclipse them from the system. Indeed, they would behave differently from untrusted nodes, and monitoring their messages would make them easily identifiable. For this reason, trusted nodes send pull requests in the same way as untrusted nodes and remain hidden from the sight of an eavesdropping adversary.

Nonetheless, they can integrate an additional defense mechanism to protect their views from view-poisoning attacks without revealing themselves to the adversary. At the end of each round, they can ignore part of the pulled IDs from untrusted nodes by not passing them to the BRAHMS sampling component and by ignoring them during the renewal of the pulled $\beta \times l_1$ entries of the node's dynamic view V.

We refer to the proportion of ignored pulled IDs as the *eviction rate*. It can be a fixed value (between 0 and 100%) for the entire system. A 100% eviction rate allows the peer sampling service to construct views as if the trusted nodes were not issuing any pull requests. The eviction rate can also be adaptive and local for each trusted node. That is, its value is set according to the proportion of trusted nodes with which the trusted node has exchanged identifiers during the current round. The intuition behind adapting the eviction rate value is that the greater the share of trusted nodes contacted during the current round, the greater the number of IDs received from trusted nodes, and the fewer the attempts to poison their views from pull responses from Byzantine nodes. If a trusted node exchange messages with many other trusted nodes in a round, then it will evict a smaller share of IDs received from untrusted nodes compared to when it contacts a few, if any, trusted nodes. We establish the following adaptive rule to determine the value of the eviction rate. First, we limit its value between 20%, when the proportion of trusted communications is above 80%, and 80%, when it is below 20%. A linear function governs the value of the eviction rate within these two limits.

Although evicting IDs from pull responses leads to fewer poisoned views of trusted nodes compared to non-Byzantine untrusted ones, it also helps an attacker identify trusted nodes. Indeed, Byzantine nodes can compare the composition of pull responses from the nodes they contact and isolate the responses that contain the fewest Byzantine IDs. We assess this identification risk in section VI.

V. EXPERIMENTAL EVALUATION

Our evaluation aims to answer the following research questions: (1) By how much does RAPTEE increase resilience to Byzantine faults compared to BRAHMS? (2) How much does RAPTEE impact the performance of BRAHMS in terms of convergence time to system discovery and to view stability?

The implementation of RAPTEE used for our experiments consists of 1,200 lines of C++ code, shared between trusted and untrusted nodes. The code that runs inside trusted nodes uses the Intel SGX SDK[1] and amounts to about 800 additional lines of C++ code. Cryptographic operations use Intel's OpenSSL SGX port[2], using RSA for asymmetric encryption and the AES-CTR mode for symmetric encryption. The code that runs on untrusted nodes features about 300 additional lines of C++. Since we do not have access to a large infrastructure supporting SGX-capable devices, we first perform a micro-benchmark to evaluate the overhead of running the code of RAPTEE's trusted node in a real SGX environment compared to an emulated SGX environment. It enables us next to calibrate our emulated SGX environment to conduct a representative large-scale experiment involving 10,000 nodes in the Grid 5000 testbed.

A. Overhead of using SGX nodes

Our first set of experiments are performed on a cluster of 40 physical machines. Each machine is an Intel Next Unit of Computing (NUC) Kit with a 2-core 3.50 GHz Intel i7

[1]Intel SGX SDK. https://software.intel.com/en-us/sgx/sdk.
[2]Intel SGX SSL. https://github.com/intel/intel-sgx-ssl.

Authorized licensed use limited to: Universite Picardie Jules Verne. Downloaded on November 15,2022 at 15:35:13 UTC from IEEE Xplore. Restrictions apply.

TABLE I: SGX performance overhead (in CPU cycles)

Peer sampling function	Standard	SGX	Mean overhead	Standard deviation
Pull request	15,623	18,593	2,970	3 %
Push message	7,521	9,182	1,661	3 %
Trusted communications	9,845	11,516	1,671	3 %
Sample list comput.	13,024	15,364	2,340	4 %
Dynamic view comput.	12,457	15,076	2,619	2 %

processor and 32 GB of RAM[3]. We consider a testbed of 200 nodes, with five nodes per machine. We instrumented the RAPTEE's code that runs inside the trusted nodes to measure the CPU-cycle consumption of each of the following functions: pull requests, push messages, trusted communication, sample list computation, dynamic view computation. We also implemented a modified version of the trusted code that emulates SGX and runs on non-capable SGX devices. We then compare the execution time of these functions on both capable and non-capable SGX devices.

We run two sets of experiments. In the first set, we deploy 100 untrusted nodes and 100 trusted nodes whose code actually uses SGX capabilities. In the second set, the 100 trusted nodes use the emulated version of the code instead. To start the experiment, each node initiates RAPTEE with a view composed of a uniform random sample of the global membership. A single experiment lasts for 200 rounds of 2.5 seconds each. We empirically set the number of rounds to 200 to ensure protocol convergence. We repeat each experiment 100 times and collect measurements at each iteration. We then compare the running time of operations on the emulated SGX nodes and on the real SGX ones, and compute the associated overhead. Table I shows these running times together with the mean and standard deviation of the CPU-cycle overhead for the functions under consideration. We use this overhead to calibrate our emulated implementation in the rest of our experiments.

B. Large-scale experimental setup

We emulate the behavior of SGX-capable nodes in a larger testbed hosted on Grid 5000. We increase the latency of each function that needs to be executed in SGX by adding a random delay that depends on the mean CPU-cycle overhead and follows its standard deviation, based on the data in Table I. Our emulated testbed consists of 10,000 nodes, running on 20 machines, each with two 16-core CPUs and 192 GB of RAM. The use of 10,000 nodes reflects approximately the sizes of the Tor ($\sim$ 8,000 relays) and Bitcoin ($\sim$ 15,000 nodes) networks.

We evaluate the resilience of RAPTEE and compare it to the nominal value of BRAHMS (see Section II). Resilience improvement is defined as the percentage drop in the number of Byzantine identifiers in the views of correct nodes. We also evaluate the performance of RAPTEE compared to BRAHMS on two other indicators: system-discovery time and view-stability time. As in the case of BRAHMS (Section II), system-discovery time is defined as the number of rounds required for all nodes to discover at least 75% of non-Byzantine IDs. View-stability time is defined as the number of rounds necessary for all non-Byzantine node views to be polluted within 10% of the average proportion of Byzantine IDs in the views of non-Byzantine nodes. We measure the performance overhead in terms of a percentage of additional rounds.

Each node runs an experiment for 200 rounds of 2.5-seconds each, during which it records the composition of its view in terms of Byzantine and honest IDs. We repeat each experiment 10 times to consolidate the results. An experimental setup consists of selected proportions of Byzantine nodes, f, and trusted nodes, t, and a fixed Byzantine eviction rate. We vary f from 10% to 30% with a step of 2%, t from 1% to 50%, and the eviction rate from 0% to 100%. We also consider the adaptive eviction rate mechanism which does not require setting a specific value. While honest nodes follow the protocol, Byzantine nodes push faulty IDs to correct nodes and always return faulty IDs to pull requests. We set the view size to 200 entries.

Figures 5 to 8 show the results of our experiments for different configurations of the Byzantine eviction rate. Their corresponding subfigures *a*, *b*, and *c* represent resilience improvements, system-discovery overhead, and view-stability overhead as a function of the proportions of Byzantine and trusted nodes, respectively. Finally, Figure 9 shows the benefits of relying on our adaptive eviction rate mechanism.

C. Resilience improvements

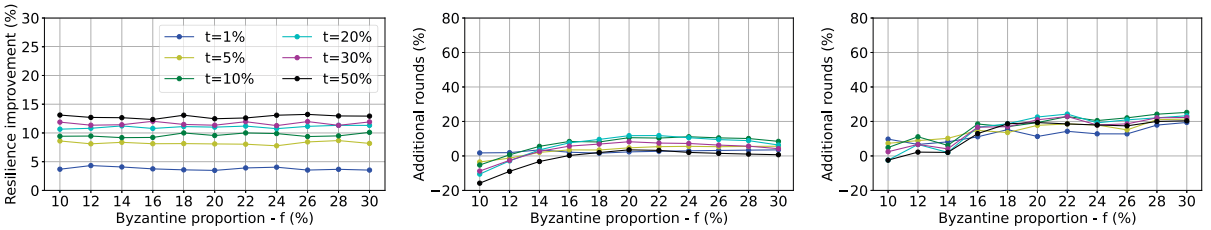
As shown in Figures 5a, 6a, and 7a, a minimum of 1% of trusted nodes ($t = 1\%$) allows RAPTEE to decrease the proportion of Byzantine IDs in the views of honest nodes by 4% under a 0% eviction rate and by 7% under a 60% eviction rate The incremental increase in t up to 50% provides further improvements in resilience, although sublinearly, up to 15%. Furthermore, the proportion f of Byzantine nodes has no impact on resilience improvements.

Considering an eviction rate of 100% might be a good idea at first glance. Indeed, for as few as 10% of Byzantine nodes, Figure 8a shows that this configuration outperforms the previous ones, providing resilience improvements ranging from 11% to 21%. However, as soon as f increases, the improvements drop, reaching between 0 and 5% percent with a Byzantine ratio of 30%, highlighting the questionable aspect of increasing the eviction rate to such an extent. A 100% eviction rate delays the process by which trusted nodes learn about other trusted nodes since only pushed IDs from the BRAHMS protocol can enter their views. This slowdown favors Byzantine nodes because they can pollute the views of honest nodes faster than trusted nodes could meet.

As shown in Figure 9a, the curves of the adaptive eviction rate have a similar shape to the 100%-eviction-rate case. However, the resilience improvements are better than any of the fixed rate scenarios described above when the fraction of Byzantine nodes is below 26%. With only 10% of Byzantine nodes, RAPTEE offers resilience improvements of between 18% and 25% over BRAHMS.

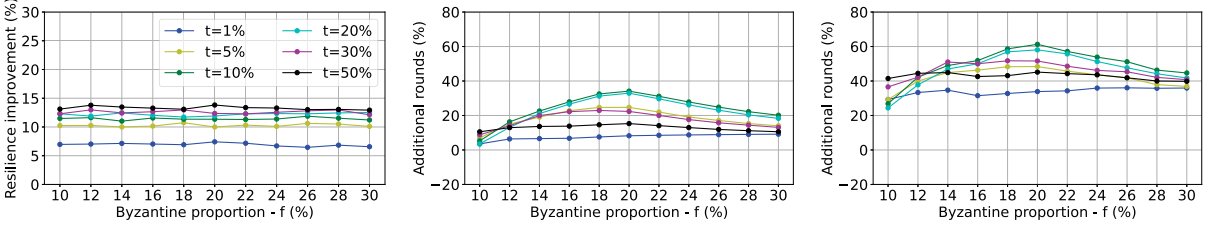
[3]One of the models recommended by Intel for experimenting with SGX.

Authorized licensed use limited to: Universite Picardie Jules Verne. Downloaded on November 15,2022 at 15:35:13 UTC from IEEE Xplore. Restrictions apply.



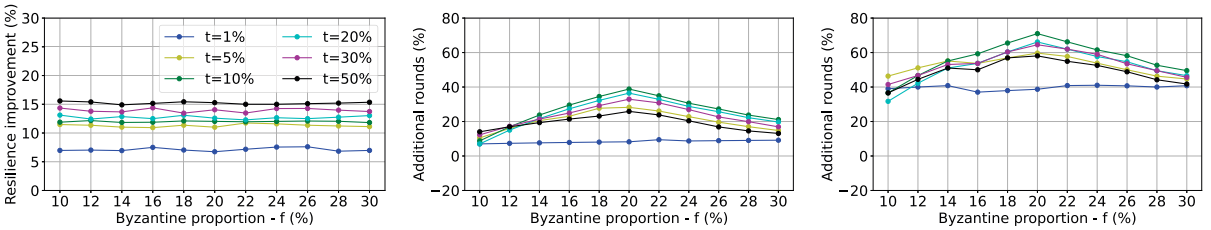
(a) Byzantine resilience gain (b) Round overhead for system discovery (%) (c) Round overhead to reach view stability (%)

Fig. 5: Resilience improvement and performance overhead under a 0% eviction rate



(a) Byzantine resilience gain (b) Round overhead for system discovery (%) (c) Round overhead to reach view stability (%)

Fig. 6: Resilience improvement and performance overhead under a 40% eviction rate



(a) Byzantine resilience gain (b) Round overhead for system discovery (%) (c) Round overhead to reach view stability (%)

Fig. 7: Resilience improvement and performance overhead under a 60% eviction rate

D. Performance overhead

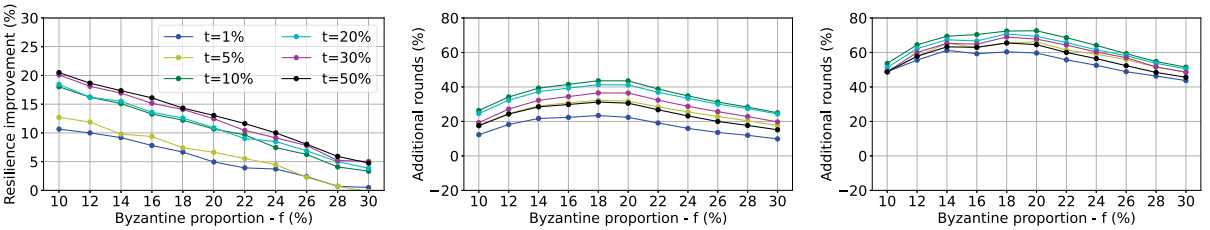
The overhead in terms of system discovery and view stability of RAPTEE increases when the eviction rate increases. However, relatively high eviction rates provide better protection against view poisoning. The adaptive eviction-rate configuration provides optimal results with overhead close to that of the 0%-eviction-rate configuration as well as the highest level of resilience. Increasing t drives up the overhead to a tipping point when larger shares of trusted communication compensate for the propagation of Byzantine IDs that slow down the discovery of non-Byzantine nodes. The overhead values of system discovery and view stability are closely related, and we can draw similar trends and interpretations. For this reason, we only comment on the system-discovery overhead in the remainder of this section.

Figure 5b shows the system-discovery overhead of RAPTEE with a 0% eviction rate. This overhead is negligible and does not exceed 3% when the number of trusted nodes is very small ($t = 1\%$). Indeed, in this configuration, trusted nodes can hardly find their siblings in the mass, making trusted communication rare and making it unlikely that malicious nodes will send Byzantine IDs to trusted nodes. Moreover, this overhead remains constant regardless of the proportion of Byzantine nodes. In other words, since trusted nodes do not evict any ID from pull responses, discovery time is barely affected by Byzantine attempts to pollute views compared with our baseline results with BRAHMS.

The system-discovery time behaves differently as the share of trusted nodes increases. When f is less than 14-16%, the time required for system discovery is actually lower than in the case of BRAHMS, with a maximum gain of 18% when considering 50% of trusted nodes. With an eviction rate of 0%, trusted nodes can quickly discover their siblings, leaving the field open for trusted communication to take over with efficient dissemination of IDs, sending them back to the honest nodes that run BRAHMS. Nonetheless, when f exceeds 14-16%, attacks by Byzantine nodes to poison the views of honest nodes hinder the benefits of efficient trusted communication, and the protocol suffers an overhead of 3-16% depending on the shares of trusted nodes considered.

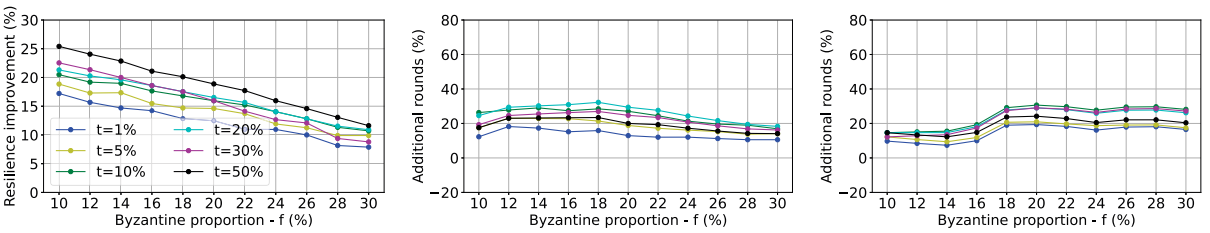
A particular pattern of discovery time should be noted. For f greater than or equal to 16%, the most extreme values of t lead

Authorized licensed use limited to: Universite Picardie Jules Verne. Downloaded on November 15,2022 at 15:35:13 UTC from IEEE Xplore. Restrictions apply.



(a) Byzantine resilience gain (b) Round overhead for system discovery (%) (c) Round overhead to reach view stability (%)

Fig. 8: Resilience improvement and performance overhead under a 100% eviction rate



(a) Byzantine resilience gain (b) Round overhead for system discovery (%) (c) Round overhead to reach view stability (%)

Fig. 9: Resilience improvement and performance overhead under the adaptive eviction rate policy

to minimal overhead. Conversely, intermediate values (*i.e.*, $t = 10$ and $t = 20$) incur the highest overhead. This bell-shaped pattern results from trusted nodes' having Byzantine identifiers in their views that are advertised by malicious nodes and their sharing them with other trusted nodes. As a result, this rapid dissemination of Byzantine identifiers slows the discovery time of the entire system, which relatively small shares of trusted nodes (*i.e.*, small values of t) cannot compensate for. A tipping point is reached once t exceeds 20%. Overhead decreases, resulting in a higher proportion of trusted nodes broadcasting more non-Byzantine identifiers.

A similar but amplified bell curve is observed in Figures 6b and 7b where the eviction rate reaches 40% and 60% respectively. This exacerbated effect results from the fact that the total number of evicted identifiers increases as t increases from 1 to 10-20%, resulting in an additional delay in the discovery of the overall system which again is not compensated for by the relatively small share of trusted nodes.

Another trend is visible on all eviction-rate configurations, except for 0% (Figures 6b, 7b, 8b, 9b). Overhead costs decrease as the proportion of Byzantines increases from less to more than 18-20%. Indeed, increasing f guides both the amount of non-Byzantine nodes and the amount of over-advertised IDs to non-Byzantine nodes. Therefore, fewer non-Byzantine identifiers and more over-advertised IDs have to be discovered, resulting in faster system discovery compared to lower values of f.

The adaptive eviction-rate mechanism provides an optimal trade-off between performance overhead and resilience. With t as small as 1%, resilience is improved by 8-18%; system discovery and view stability require only 15-20% more rounds.

VI. Security analysis

This section provides a security analysis of RAPTEE against two attack vectors: trusted node identification, and view-poisoned trusted node injection.

A. Trusted node re-identification

In the original BRAHMS paper [6], the authors prove that an adversary that tries to isolate nodes by launching targeted attacks on them cannot partition the network. However, RAPTEE's use of trusted communications makes trusted nodes a particularly attractive target for the adversary. If the adversary could identify these nodes, it could launch a targeted attack to pollute their views with Byzantine IDs. The adversary would thus have a backdoor to inject Byzantine IDs into the protocol to increase the dissemination speed of Byzantine IDs. This could inhibit the benefits of trusted communications on resilience to Byzantine behaviors and could further increase the time required to reach stability and to discover new peers. To evaluate this attack feasibility, we must distinguish between two cases: after reaching view stability and before.

We recall that view stability is defined as the state in which the maximum difference in view composition between any non-Byzantine node and the average of all non-Byzantine nodes does not exceed 10%. Our experiments show that once this state is reached, the difference in view composition between trusted and untrusted nodes does not exceed 2% on average. This difference is therefore much smaller than the 10% threshold used to define view stability. This makes it very difficult for an adversary to identify trusted nodes even with global knowledge of all views in the system.

On the other hand, before reaching view stability, trusted nodes propagate their views to untrusted nodes via answers

Authorized licensed use limited to: Universite Picardie Jules Verne. Downloaded on November 15,2022 at 15:35:13 UTC from IEEE Xplore. Restrictions apply.

to pull requests, which are sometimes (and purposely) significantly different from the views of untrusted nodes. To assess the extent to which this can lead to successful identification, we consider an attack in which each Byzantine node measures the proportion of Byzantine IDs in the pull answers it receives from each non-Byzantine node and provides this information to the adversary. The adversary first computes the average percentage of Byzantine IDs in the views of all honest nodes. Then, for each honest node, it computes the difference between this average percentage and the percentage of Byzantine IDs in the honest node's view. If the difference exceeds the threshold, the adversary labels the node as a trusted node. We experimentally tested several thresholds and the one that maximizes the identification outcome is 10%.

We evaluate the effectiveness of this attack with 10% of Byzantine nodes in Figures 10 and 30% of Byzantine nodes in Figure 11. For each Byzantine configuration, we present recall (subfigures (a)), precision (subfigures (b)), and F1-score (subfigures (c)) values for different values of eviction-rate (denoted ER in the figures) and shares of trusted node. Figure 12 highlights the results with an adaptive eviction rate.

Results show that the effectiveness of the attack grows steadily with the percentage of trusted nodes in the system. It is clear that the more nodes of trust there are, the easier it is for the adversary to identify some of them. However, success rate also depends on the eviction rate. In particular, Figure 10 and Figure 11 show an increase in recall of 0.14 (14%) when the eviction rate is increased from 0 to 100%, and an increase in precision of about 0.12 under the same conditions. An eviction rate of 60%, for example, leads to an identification precision of 50% and a recall of about 30% with 20% of trusted nodes. Moreover, these values increase only slightly with the percentage of Byzantine nodes.

Although the above values may seem high, they are drastically reduced by adopting an adaptive eviction rate. Figure 12 shows that precision varies between 0 and 0.3 when going from 1 to 50% of trusted nodes and does not significantly depend on the proportion of Byzantine nodes. Recall follows a similar pattern, although with a stronger dependence on the percentage of Byzantine nodes. Its values range from 0.01 with 10% of Byzantine nodes, and 0.30 with 30% of Byzantine nodes.

B. View-poisoned trusted node injection

We have shown that trying to identify trusted nodes in order to isolate them bring only limited results for the adversary, especially with an adaptive eviction rate. As an alternative, the adversary can adopt a totally different strategy by purchasing SGX-capable devices and using them as a means of disseminating Byzantine node identifiers to real trusted nodes.

Specifically, the adversary can deploy some SGX-capable nodes in a network that contains only Byzantine nodes, in order to fill their views with Byzantine identifiers. The adversary can then move these view-poisoned trusted nodes into the actual network and wait for them to disseminate the Byzantine identifiers to the actual trusted nodes.

Figure 13 shows the effect of this attack in a network with different initial proportions of trusted nodes, $t = 1\%, 10\%$, and 30%. Each plot shows how the resilience-improvement metric varies as a function of the proportion of Byzantine nodes with several percentages of view-poisoned trusted nodes added by the adversary 1%, 5%, 10%, 20%, and 30%. In all the plots, the black line represents the baseline with t honest trusted nodes and no attacks.

When the proportion of honest trusted nodes is small, $t = 1\%$, and for a relatively small proportion of Byzantine nodes, the addition of view-poisoned trusted nodes does not appear to significantly harm resilience. Figure 13a shows that adding a large number of view-poisoned trusted nodes even seems to improve resilience. Indeed, even view-poisoned nodes are forced to work with correct BRAHMS implementations. As a result, all trusted nodes, including view-poisoned ones, end up removing the overrepresented identifiers, and the view-poisoned nodes end up reinforcing the trusted portion of the network. As the proportion of Byzantine nodes increases (to the right of the plot in Figure 13a), the sampling process becomes less effective at removing Byzantine identifiers from the views, thus decreasing the percentage resilience improvement.

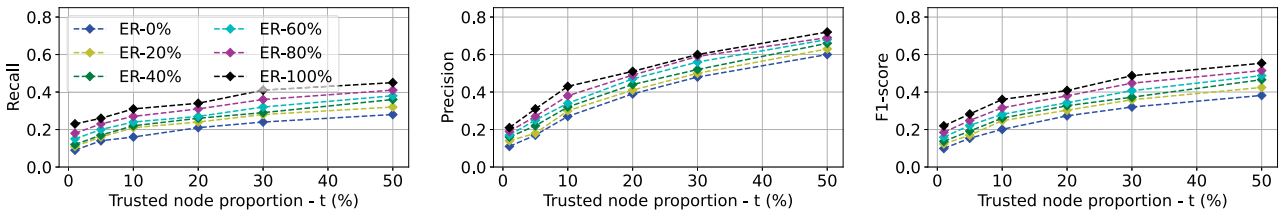
As the proportion of honest trusted nodes increases from $t = 10\%$ in Figure 13b to $t = 30\%$ in Figure 13c, the benefit of having additional trusted nodes with a small portion of Byzantine nodes disappears as adding a fraction of trusted nodes to a larger set makes less of a difference and instead becomes significantly counterproductive, when $t = 30\%$.

VII. DISCUSSION

Our experimental evaluation in Section V shows that RAPTEE can provide significant resilience improvement over BRAHMS. Its adaptive eviction strategy provides a 17% resilience improvement in the presence of 10% of Byzantine nodes at the minimal cost of 1% of trusted nodes. Even when the proportion of Byzantine nodes increases to 30%, RAPTEE still provides an 8% improvement.

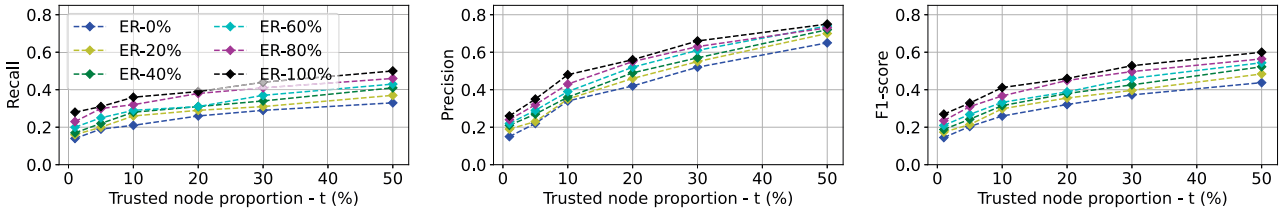
Most importantly, our analysis in Section VI also shows that RAPTEE with 1% of trusted nodes effectively resists attackers that can (i) try to identify the trusted nodes in order to isolate them, or (ii) inject view-poisoned trusted nodes that try to spread malicious identifiers to honest nodes. With respect to (i), an attacker can identify only less than 10% of the trusted nodes with precision below 2% for an F1-score of less than 3%. When it is possible to increase the proportion of honest trusted nodes to 10%, RAPTEE provides even higher levels of resilience improvement above 20%, with trusted nodes becoming only slightly more detectable (precision and F1-score below 15%). With respect to (ii), an attack with 5% of view-poisoned trusted nodes in a system with 1% of honest trusted nodes even improves resilience to Byzantine nodes when the latter's proportion is not too high (below 20%). Together these results suggest that RAPTEE can serve as an effective overlay for a number of Byzantine-sensitive applications like smart-contract platforms and cryptocurrencies.

Authorized licensed use limited to: Universite Picardie Jules Verne. Downloaded on November 15,2022 at 15:35:13 UTC from IEEE Xplore. Restrictions apply.



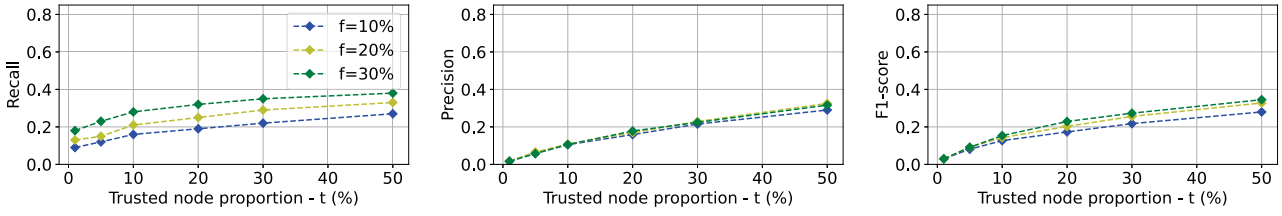
(a) Recall (b) Precision (c) F1-score

Fig. 10: Precision, recall and F1-score of trusted-node identification under 10% of Byzantine nodes



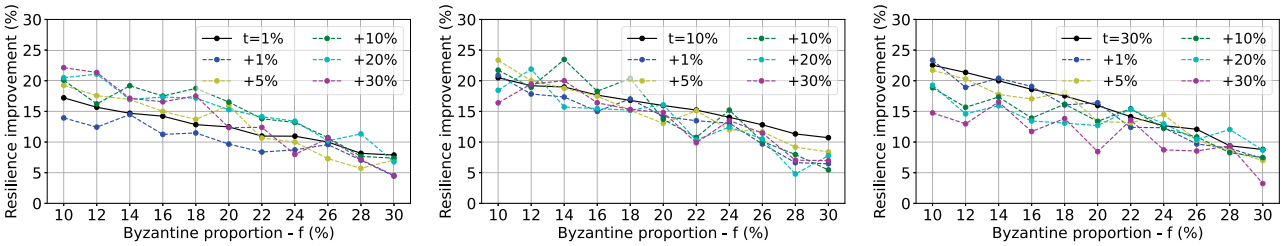
(a) Recall (b) Precision (c) F1-score

Fig. 11: Precision, recall and F1-score of trusted-node identification under 30% of Byzantine nodes



(a) Identification recall (b) Identification precision (c) Identification F1-score

Fig. 12: Precision, recall and F1-score of trusted-node identification under adaptive eviction rate



(a) Attack on a system with $t = 1\%$ (b) Attack on a system with $t = 10\%$ (c) Attack on a system with $t = 30\%$

Fig. 13: Corrupted trusted node injection

Support from Intel SGX on personal computers in the future may be unclear. In light of the advent of side channel attacks requiring physical access or process co-location, it is possible that SGX might be available only on server-grade CPUs in the future. This could impact the proportion of SGX nodes that can participate in RAPTEE. We note, however, that even a small number of SGX nodes can provide significant improvement, mitigating the impact of this limitation.

VIII. RELATED WORK

Random peer sampling (RPS) has mostly been studied in non-adversarial environments. Cyclon [21], Newscast [20], and the more generic protocol framework for Gossip based peer sampling of Jelasity *et al.* [15] efficiently handle churn and quickly discover other nodes in the network. Our contribution uses such a RPS protocol between trusted nodes. However, these protocols are easily disturbed by Byzantine nodes performing eclipse or poisoning attacks.

A few studies targeted the problem of making RPS tolerant to Byzantine nodes [2], [3], [6], [16]. In the *Secure peer Sampling* framework [16], each node uses a detection mechanism to identify and blacklist maliciously acting nodes. This protocol remains, however, vulnerable to rapid flooding attack

Authorized licensed use limited to: Universite Picardie Jules Verne. Downloaded on November 15,2022 at 15:35:13 UTC from IEEE Xplore. Restrictions apply.

as correct nodes cannot identify and blacklist attackers before being overwhelmed by them and isolated. RAPTEE reduces for trusted nodes the risk of isolation through such attacks by dropping pull requests answers from untrusted nodes.

Instead of using a detection mechanism, Brahms [6] employs push limiting and view sampling in order to mitigate the over-representation of malicious nodes in the view of honest ones. The use of a Trusted Execution Environment in RAPTEE enables better performance than Brahms in terms of resilience against malicious nodes and alleviates some of the significant running time overhead of Brahms.

Anceaume *et al.* [1] formally analyze the requirements of peer sampling protocols when these need to resist malicious attacks. They observe that a necessary condition lies in limiting the request rates of nodes. RAPTEE complements this measure with its eviction-rate policy. The same authors [2], [3] employ count-min sketches to unbias a biased stream of identifiers. Adopting a similar technique in RAPTEE could constitute interesting future work.

With respect to resistance to Sybil attacks, HAPS [7] uses a prefix tree to limit the number of known identifiers in each IP-address prefix. This effectively limits Sybil attacks under the assumption that honest nodes are uniformly distributed over the IP address space. This idea is orthogonal to our rate-limiting mechanism and it could constitute an interesting addition to RAPTEE a large-scale deployment.

Finally, GossipSub [23] is a recent proposal for strengthening the gossip-based dissemination in the Ethereum 2.0 and Filecoin networks. These open blockchain systems rely on gossip to quickly propagate of transactions and blocks between miners. GossipSub implements an ad hoc mesh construction protocol that maintains a balanced in- and out degree for nodes, as do protocols from the peer sampling family. This protocol does not, however, target diversity, i.e., the constant renewal of peers in the views of each node. This makes it specific for dissemination and unfit for other classes of gossip-based protocols such as overlay construction [14], [22] or self-organizing aggregation [13]. GossipSub reduces the risk of attacks by implementing a reputation mechanism. Every node ranks peers in its view, and peers are incentivized to follow the protocol for a long period of time and with the same communication partners.

IX. CONCLUSION

We presented RAPTEE, a novel Byzantine-tolerant random peer sampling protocol that builds and leverages trusted gossip-based communication. RAPTEE interoperates with BRAHMS, the most resilient peer sampling protocol to date, and reduces the impact of a poisoning attack against its nodes. Our experiments show that with only 1% of SGX-capable devices, RAPTEE can reduce the proportion of Byzantine IDs in the views of honest nodes by up to 17% when the system contains 10% of Byzantine nodes, at the cost of very limited overhead. In addition, the security guarantees of RAPTEE hold even in the presence of a powerful attacker attempting to identify trusted nodes and injecting corrupted trusted nodes.

REFERENCES

[1] E. Anceaume, Y. Busnel, and S. Gambs, "Characterizing the Adversarial Power in Uniform and Ergodic Node Sampling," in *AlMoDEP '11 (colocated with DISC 2011)*. Rome, Italy: ACM, Sep. 2011. [Online]. Available: https://hal.inria.fr/inria-00617866

[2] ——, "On the power of the adversary to solve the node sampling problem," *Transactions on large-scale data-and knowledge-centered systems*, vol. 11, 2013.

[3] E. Anceaume, Y. Busnel, and B. Sericola, "Uniform node sampling service robust against collusions of malicious nodes," in *43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, ser. DSN. IEEE, 2013.

[4] O. Babaoglu, G. Canright, A. Deutsch, G. A. D. Caro, F. Ducatelle, L. M. Gambardella, N. Ganguly, M. Jelasity, R. Montemanni, A. Montresor *et al.*, "Design patterns from biology for distributed computing," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 1, no. 1, pp. 26–66, 2006.

[5] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lanteri, J. Leduc, N. Melab *et al.*, "Grid'5000: a large scale and highly reconfigurable experimental grid testbed," *The International Journal of High Performance Computing Applications*, vol. 20, no. 4, pp. 481–494, 2006.

[6] E. Bortnikov, M. Gurevich, I. Keidar, G. Kliot, and A. Shraer, "Brahms: Byzantine resilient random membership sampling," *Computer Networks*, vol. 53, no. 13, pp. 2340–2359, 2009.

[7] A. Bouchra Pilet, D. Frey, and F. Taiani, "Foiling Sybils with HAPS in permissionless systems: An address-based Peer Sampling Service," in *IEEE Symposium on Computers and Communications*. IEEE, 2020.

[8] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, "Min-Wise Independent Permutations," *Journal of Computer and System Sciences*, vol. 60, no. 3, pp. 630–659, Jun. 2000.

[9] V. Costan and S. Devadas, "Intel SGX Explained," Tech. Rep. 086, 2016.

[10] J. R. Douceur, "The Sybil Attack," in *Peer-to-Peer Systems*, ser. Lecture Notes in Computer Science, P. Druschel, F. Kaashoek, and A. Rowstron, Eds. Berlin, Heidelberg: Springer, 2002, pp. 251–260.

[11] P. T. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec, "Lightweight probabilistic broadcast," *ACM Transactions on Computer Systems (TOCS)*, vol. 21, no. 4, pp. 341–374, 2003.

[12] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse Attacks on Bitcoin's Peer-to-Peer Network," in *24th USENIX Security Symposium*, 2015, pp. 129–144.

[13] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Transactions on Computer Systems (TOCS)*, vol. 23, no. 3, pp. 219–252, 2005.

[14] ——, "T-man: Gossip-based fast overlay topology construction," *Computer networks*, vol. 53, no. 13, pp. 2321–2339, 2009.

[15] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen, "Gossip-based peer sampling," *ACM Trans. Comput. Syst.*, vol. 25, no. 3, pp. 8–es, Aug. 2007.

[16] G. P. Jesi, A. Montresor, and M. van Steen, "Secure peer sampling," *Computer Networks*, vol. 54, no. 12, pp. 2086–2098, 2010.

[17] M. Matos, V. Schiavoni, P. Felber, R. Oliveira, and E. Riviere, "Lightweight, efficient, robust epidemic dissemination," *Journal of Parallel and Distributed Computing*, vol. 73, no. 7, pp. 987–999, 2013.

[18] S. Pinto and N. Santos, "Demystifying ARM TrustZone: A comprehensive survey," *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, pp. 1–36, 2019.

[19] E. Riviere and S. Voulgaris, "Gossip-based networking for internet-scale distributed systems," in *International Conference on E-Technologies*. Springer, 2011, pp. 253–284.

[20] N. Tölgyesi and M. Jelasity, "Adaptive peer sampling with Newscast," in *European Conference on Parallel Processing*, ser. EuroPar. Springer, 2009, pp. 523–534.

[21] S. Voulgaris, D. Gavidia, and M. Van Steen, "Cyclon: Inexpensive membership management for unstructured p2p overlays," *Journal of Network and systems Management*, vol. 13, no. 2, pp. 197–217, 2005.

[22] S. Voulgaris and M. van Steen, "VICINITY: A Pinch of Randomness Brings out the Structure," in *Middleware 2013*. Berlin, Heidelberg: Springer, 2013, pp. 21–40.

[23] D. Vyzovitis, Y. Napora, D. McCormick, D. Dias, and Y. Psaras, "GossipSub: Attack-resilient message propagation in the Filecoin and Eth2.0 networks," *arXiv preprint arXiv:2007.02754*, 2020.

Authorized licensed use limited to: Universite Picardie Jules Verne. Downloaded on November 15,2022 at 15:35:13 UTC from IEEE Xplore. Restrictions apply.