On Detecting Route Hijacking Attack in Opportunistic Mobile Networks

Ala Altaweel, Radu Stoleru, Guofei Gu, Arnab Kumar Maity, Suman Bhunia

Abstract—In this paper, we show that Hybrid Routing and Prophet protocols in Opportunistic Mobile Networks (OMNs) are vulnerable to the *CollusiveHijack* attack, in which a malicious attacker, Eve, compromises a set of nodes and lies about their Inter-Contact-Times (ICTs). Eve claims that her nodes meet more frequently than in reality to hijack the routes of legitimate nodes in OMNs. The CollusiveHijack attack enables Eve to launch more severe attacks like packet modification, traffic analysis, and incentive seeking attacks. To identify the CollusiveHijack attack, we propose the Kolmogorov-Smirnov two-sample test to determine whether the statistical distribution of the packets' delays follows the derived distribution from the ICTs among the nodes. We propose three techniques to detect the CollusiveHijack attack, the Path Detection Technique (PDT), the Hop Detection Technique (HDT), and the Early Hop Detection Technique (EHDT), which trade off compatibility with the Bundle Security Protocol, the detection rate, and the detection latency. We evaluated our techniques through extensive trace-driven simulations and a proof-of-concept system implementation and show that they can detect CollusiveHijack attacks with 80.0% to 99.4% detection rates (when Eve hijacks more than 60 packets) while maintaining a low false positive rate (∼3.6%) and a short detection latency (7-14 hours) for EHDT (75%-85% enhancement compared to PDT and HDT).

1 Introduction

Opportunistic Mobile Networks (OMNs) refer to wireless networks in which mobile nodes are intermittently connected, without stable end-to-end paths. OMNs have a wide range of applications, e.g., disaster response [1], [2], battlefield communications, social networks applications (Firechat and 1am [3] [4]), aerospace [5], railways [6], vehicular networks [7], [8], and IoT-based mobile health networks [9]. Due to the intermittent connectivity between the nodes in OMNs, their routing protocols (e.g., Hybrid Routing (HRP) [10] and Prophet Protocols [11] [12]) adopt the store-carry-and-forward mechanism to deliver the packets. HRP and Prophet protocols learn from the nodes' past contact history in order to make forwarding decisions. That is, a pair of nodes in OMNs measures their past contact frequencies to find the probability of their future contacts. The Inter-Contact-Time (ICT), which is the time duration between two contacts between a pair of nodes, is used to measure the contact frequency and the delivery capability of the nodes [13] [14]. The ICTs that were extracted from OMNs' real-world mobility traces [15], [16] show that they might be at the scale of seconds, minutes, or even days.

In this paper, we show that one major security concern and research challenge for HRP and Prophet protocols is their vulnerability to a Route Hijacking attack. In this attack, that we called *CollusiveHijack*, a malicious attacker, Eve, compromises a set of nodes and lies about their ICTs. Eve claims that her nodes meet more frequently than in reality, in order to deceive HRP and Prophet protocols. Eve aims to hijack the packets of the legitimate nodes in OMNs. We found that the CollusiveHijack attack can be launched due to the fact that nodes in OMNs have no way of verifying whether the claimed ICT between a pair of nodes is true or false, even if the claimed ICTs are signed. That is, Eve can successfully launch the CollusiveHijack attack since her nodes share their private keys and sign their claimed ICTs. The CollusiveHijack enables Eve to launch more severe attacks like: a) packet modification attack [17], which enables Eve to corrupt the contents of the packets, thus enforcing packet re-transmissions and a decrease in the packet delivery ratio [18] (i.e., waste of network resources, power, bandwidth); b) eavesdropping and traffic analysis attack, which enables Eve to identify the types of network traffic and apps of the legitimate nodes [19] [20] in OMNs; c) incentives seeking attack, that is, if an incentive based mechanism [21] is employed in OMNs, Eve's nodes can deliver the hijacked packets to get more credit and higher reputation.

To the best of our knowledge, a route hijacking attack has neither been identified nor addressed in OMNs as most of the previous research in these networks addressed jamming, blackhole, flood, wormhole, and packet dropping attacks [22]–[28] or focused on authentication, trust-management, and privacy-preserving [29]–[35]. However, many approaches have been proposed to detect route hijacking in Internet. Approaches like [36]–[40] collect BGP updates and routing tables from a public BGP monitoring infrastructure [41]–[43] and raise alarms when a change in the origin Autonomous System (AS) of a prefix or a suspicious route is observed. Other approaches [36]–[40] require network administrators (which are not available in OMNs) and a list of owned/reached IPs a priori (the nodes in OMNs do not know their future contacts). The proposed algorithms in [44] [45] continuously probe Internet to detect whether any data path changes by leveraging pings/traceroutes tools

- *A preliminary version of this work appeared in IEEE CNS 2019*
- *Dr. Ala Altaweel is with University of Sharjah, Dr. Radu Stoleru and Dr. Guofei Gu are with Texas A&M University. Dr. Arnab Kumar Maity is with Pfizer Inc. Dr. Suman Bhunia is with Miami University of Ohio.*

to monitor the connectivity of a prefix and raise an alarm when significant changes in the reachability of a prefix (or the paths leading to it) are detected. However, due to the intermittent connectivity among the nodes in OMNs, pings/traceroutes tools fail to work. Hence, the CollusiveHijack attack is still an open research problem for OMNs.

In order to address the aforementioned research challenge, we propose to detect the CollusiveHijack attack by employing the Kolmogorov-Smirnov two-sample test (KS2ST) [46], a well known test for comparing two statistical distributions. That is, it measures the distance between the empirical distribution functions of two samples to determine whether they have been drawn from the same distribution or not. The KS2ST has been used by previous works for detecting covert channels, detecting selfish wireless nodes, and in intrusion detection systems [47]–[50]. We propose three techniques to detect the CollusiveHijack attack: the Path Detection Technique (PDT), the Hop Detection Technique (HDT), and the Early Hop Detection Technique (EHDT). PDT, HDT, and EHDT offer a trade-off between the *compatibility* with the Bundle Security Protocol (BSP) [51] (if additional steps are required at the intermediate nodes) and the *detection rate* as well as the *detection latency* against the CollusiveHijack attack. In PDT, the destination performs a path-wise detection by collecting the packets' delays along the path. The intermediate nodes in the path are authenticated by leveraging the sequential authenticators capability of BSP. The destination uses the KS2ST to test whether the statistical distribution of the packets' delays follows the statistical distribution that is derived from the claimed ICTs of the intermediate nodes. In HDT, the destination performs a hop-wise detection by requiring additional information from the intermediate nodes (the packets' receiving times). The destination leverages the KS2ST to detect whether each hop in the path is compromised. Our experiments show that HDT can achieve a higher detection rate against the CollusiveHijack attack than PDT.

In order to early detect the CollusiveHijack attack at the intermediate nodes rather than delay the detection until the packets are received by the destination nodes (as is the case of both PDT and HDT), we propose the Early Hop Detection Technique(EHDT), which aims to reduce the detection latency of the attack. This research is an extension of the preliminary conference version appearing in the Proceedings of IEEE Conference on Communications and Network Security (CNS'19) [52] with the following contributions:

- We demonstrate, via implementation and extensive simulations using two real-world mobility traces, a successful CollusiveHijack attack against HRP and Prophet protocols.
- We present three techniques, the Path Detection Technique (PDT), the Hop Detection Technique (HDT), and the Early Hop Detection Technique (EHDT), to detect the CollusiveHijack attack.
- We demonstrate the feasibility of PDT, HDT, and EHDT through extensive trace-driven simulations using two real-world mobility traces and a proof-of-concept system implementation.
- We demonstrate the effectiveness of PDT, HDT, and EHDT by showing that they identify CollusiveHijack attacks with a high detection rate while maintaining a low false positive rate and with a short detection latency.

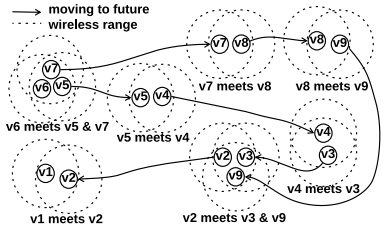
The rest of the paper is organized as follows. In Section 2 we present our network and adversary models. We motivate our research in Section 3 by launching a successful CollusiveHijack attack against HRP and Prophet protocols. The design of PDT, HDT, and EHDT is presented in Section 4. We illustrate the experimental setup of our simulation and implementation in Section 5 before presenting the evaluation results in Section 6. The security analysis of PDT, HDT, and EHDT is presented in Section 7. In Section 8, we thoroughly survey the state-of-the-art and security threats OMNs. Finally, we conclude our paper in Section 9.

2 Network and Adversary Models

This section presents the network and adversary models.

2.1 Network Model

We consider an OMN, as shown in Figure 1, with nodes that run HRP or Prophet protocol. v_1 learns from the contact records (that are received via v_2) that either (v_2, v_3, v_4, v_5) or (v_2, v_9, v_8, v_7) can deliver v_1's packets to v_6. We assume that it is possible to achieve a time synchronization between all OMN's nodes at the scale of one second (which is sufficient since the ICTs in OMNs are at the scale of seconds/minutes). In the following paragraphs, we present HRP and Prophet in more detail.



Nodes	v2-v3-v9	v3-v4	v4-v5	v5-v6-v7	v7-v8	v8-v9
True cliam	1	1	1	1	1	1
False claim	1	1	1	1	2	2

Fig. 1: Contact frequency for 1 day

Hybrid Routing Protocol (HRP) [10] [53] is a *limited* replication-based protocol that relies on the observation that path delay correlations can impact performance improvements gained from packet replication. HRP captures the potential correlation between the ICTs for different nodes and decides how much replication should be used for different network environments. HRP introduces two concepts: the *replication factor* and the *replication gain*. The replication factor is the total number of data copies created at the source for a given packet. If D_r is the random variable for routing delay when replication factor is r, then the replication gain is $E[D_1]/E[D_r]$. The replication gain captures the benefit of replication in terms of delay improvement. HRP demonstrated mathematically and experimentally that the delay correlation affects the benefit of replication and it is important to capture it to estimate the replication gain and make better routing decision. HRP is implemented as a user-space daemon service [53].

Prophet protocol [11] [12] is an *unlimited* replication-based protocol that relies on a probabilistic metric called

delivery predictability, $\Psi \in [0, 1]$. Ψ is established at each node indicating the probability of delivering a message to all other nodes. As shown in Figure 1, when v_6 encounters v_5, they exchange their Ψ's and update them accordingly. $\Psi_{(v_6,v_5)}$, which is v_6 delivery predictability for v_5, is updated according to $\Psi_{(v_6,v_5)} = \Psi_{(v_6,v_5)_{old}} + (1 - \Psi_{(v_6,v_5)_{old}}) \times \Psi_{init}$. $\Psi_{init} \in$ [0,1] is a constant to ensure that nodes that frequently meet have high Ψ's. If two nodes do not meet for a while, their Ψ's must *age*. The aging equation for v_5 and v_6 is $\Psi_{(v_6,v_5)} = \Psi_{(v_6,v_5)_{old}} \times \gamma^{Tu}$. $\gamma \in$ [0, 1) is a constant to decide how large impact the aging should have on Ψ and Tu is the number of time units that has elapsed since the last time Ψ was aged. Ψ also has a *transitive* property, as shown in Figure 1, which is based on the observation that if v_i frequently encounters v_j, and v_j frequently encounters v_k, then v_k is a good carrier to forward the messages to v_i. E.g., $\Psi_{(v_4,v_6)} = \Psi_{(v_4,v_6)_{old}} + (1 - \Psi_{(v_4,v_6)_{old}}) \times \Psi_{(v_4,v_5)} \times \Psi_{(v_5,v_6)} \times \beta$. $\beta \in$ [0, 1] is a constant to decide how large impact the transitivity should have on Ψ. When v_i, which has a message for v_k, encounters v_j, then, v_i replicates its message to v_j only if $\Psi_{(v_j,v_k)} > \Psi_{(v_i,v_k)}$. Prophet protocol is implemented as a user-space daemon service [54] [55] and is considered as the most widely used protocols for Opportunistic Mobile Network (OMN). Recently, many machine learning and contextual information based approaches have been proposed to enhance the routing performance (e.g., packet delivery ratio) and the energy consumption of Prophet [56]–[62].

In both HRP and Prophet protocols [10]–[12], the ICTs are exchanged (via small-size *control* packets) among the nodes in a broadcasting manner when the nodes encounter each other. These ICTs are used to make future routing/forwarding decisions for the large-size *data* packets. We assume that HRP and Prophet routing protocols use a public-key identity-based cryptography. Each node's private key is only known by itself to guarantee the authentication and message integrity. The nodes also use their private keys to sign the announced ICTs. For example, in Figure 1, once v_1 and v_2 encounter each other, they calculate/update and sign (using their private keys) the ICT between each other and they also exchange all ICTs information that they had previously received from other nodes.

2.2 Adversary Model

We assume that the OMN has two types of nodes (*honest* and *compromised*). We also assume that a malicious attacker, Eve, has control of the compromised nodes that are infected by malware. Eve's nodes collusively share each other's private keys and lie about their ICTs. For example, in Figure 1, v_1 has a packet for v_6 and the intermediate nodes (v_7, v_8, v_9), that are compromised, decrease their ICTs. That is, instead of informing v_1 that v_8 encountered v_9 one time during the last day, which is the truth, they claim that they encountered each other twice during the last day (both of v_8 and v_9 sign the claimed/fake ICT). The ICT between v_7 and v_8 is also decreased, as shown in Figure 1. In case of HRP, decreasing the ICTs of (v_8, v_9) and (v_7, v_8) improves their Ψ's to v_6 as HRP uses ICTs to measure nodes' delivery capabilities (i.e., HRP replicates the packets to the nodes with lower ICTs). In case of Prophet, Eve exploits the transitive and aging properties by decreasing the ICTs of its nodes to increase their Ψ's. As shown in Figure 1, decreasing the ICTs of (v_8, v_9) resets Tu to 0 and increases $\Psi_{(v_9,v_8)}$ due to the aging property. Due to the transitive property ($\Psi_{(v_9,v_6)} = \Psi_{(v_9,v_6)_{old}} + (1 - \Psi_{(v_9,v_6)_{old}}) \times \Psi_{(v_9,v_8)} \times \Psi_{(v_8,v_6)} \times \beta$), $\Psi_{(v_9,v_6)}$ also increases and becomes $> \Psi_{(v_2,v_6)}$. Hence, v_1 forwards its packets to v_8 and v_9. Based on our adversary model, we define the CollusiveHijack attack as follows:

Definition 2.1 (CollusiveHijack)**.** *In this attack, Eve, compromises a set of nodes in OMNs and lies about their Inter-Contact-Times(ICTs). Eve claims that the compromised nodes meet more frequently than in reality (by decreasing their ICTs) to hijack the data packets' routes of legitimate nodes in OMNs.*

Notice that Eve's goal is to influence the routing decision of the legitimate nodes in OMNs to hijack their *data* packets. That is, the CollusiveHijack attack does not enable Eve to hijack the *control* packets that contain the ICTs information as these packets are exchanged in a broadcasting manner in OMNs [10]–[12]. We assume that Eve is not interested in launching jamming, blackhole, flood, wormhole, or packet dropping attacks during the ICTs broadcasting phase (or other phases) in OMNs as these attacks have been addressed by previous works [22]–[28], [30]–[35].

3 Motivation

We show the impact of CollusiveHijack by conducting two set of experiments, as described in the following sections.

3.1 Launching CollusiveHijack on a real-world testbed

We deploy the HRP and Prophet's daemon services [53] [54] on 11 Asus Eee notebooks that run Ubuntu 14.04 LTS. The notebooks operate in ad-hoc mode on IEEE 802.11 channel 3 (2.4 GHz). Due to space limitation and for the ease of testing, we place all notebooks together and emulate a fully connected multi-hop mesh network by manipulating firewall configurations. To this end, we connect all notebooks to a server through Ethernet cable and issue *iptables* and *ip6tables* commands from the server to create different typologies, as shown in Figure 2(a). In order to create contact events between the notebooks as it is the case in OMNs, we conducted a dynamic on-off network experiment. First, we created the topology shown in Figure 2(b). Second, we turned each node on and off randomly (according to an exponential distribution) by issuing *iptables* and *ip6tables* commands with different on-proportion. The expected total duration for one on-off cycle is set to 60 seconds. We generate a data flow from v_1 to v_{11} with 1 KB and set its deadline to 300 seconds. We leveraged HRP's *hrptclient* and IBR-DTN's *dtnsend* and *dtnrecv* for the data flow. We used the default values of $\beta = 0.9$ and $\gamma = 0.999$ for the Prophet protocol. The duration of each experiment is 30 minutes including a warm-up period for 5 minutes. The warm-up period is used by HRP and Prophet to learn about the contact events and the ICTs before starting the data flow.

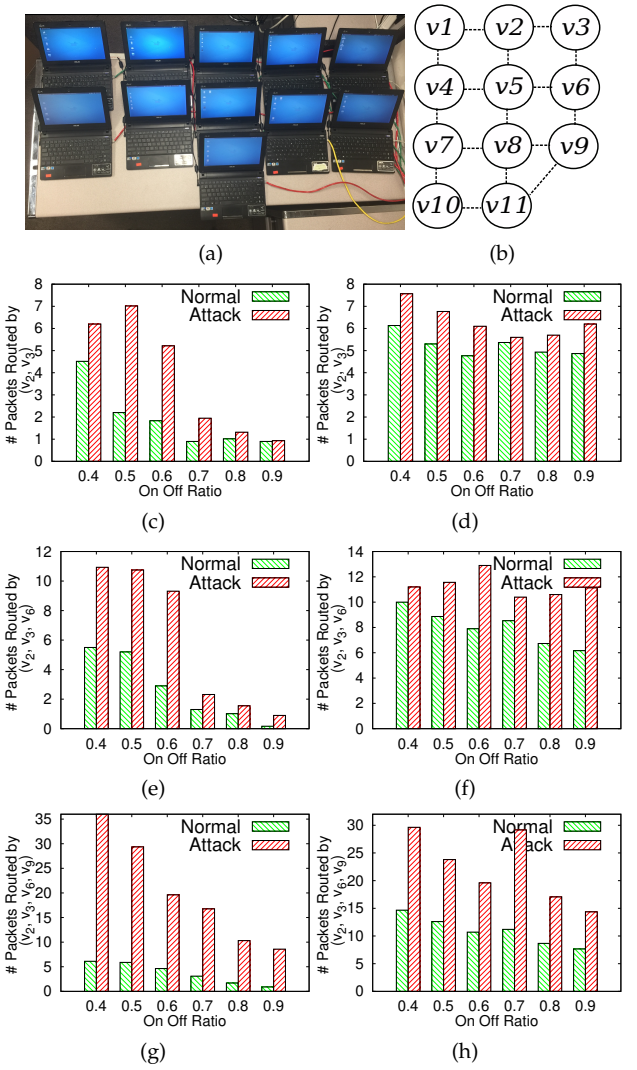


Fig. 2: (a) Attack implementation testbed. (b) Network topology. Routed packets for HRP (c, e, g) and Prophet (d, f, h).

For each experiment, we considered a normal scenario and an attack scenario. In the attack scenario, we assigned the same ICTs that were used during the normal scenario among the honest nodes, however, the ICTs of the compromised nodes are multiplied by 0.5. We intend to show that the CollusiveHijack can be successfully launched against HRP and Prophet under diverse network conditions. That is, by varying the on-proportion $\in$ [0.4, 1.0] to emulate the well-connected mesh network and the sparsely connected OMN. We tracked the routing paths via *tcpdump* to find the number of replicated packets to the compromised nodes during each experiment. We repeated each experiment 30 times (with different ICTs) and we averaged these 30 runs per each experiment.

Figures 2(c) and 2(d) show the total number of routed packets when v_2 and v_3 are compromised for HRP and Prophet, respectively. During the attack scenario of these experiments, the ICT between v_2 and v_3 is claimed to be 0.5 of the ICT between v_2 and v_3 during the normal scenario. For example, if in the normal scenario the average ICT between

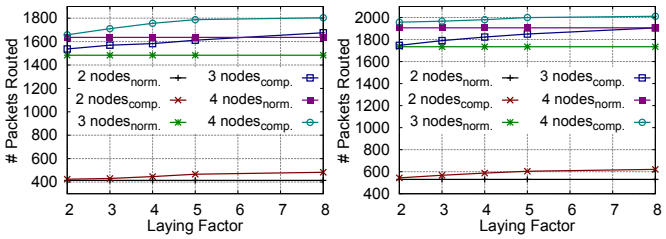


Fig. 3: Routed packets for Prophet using (a) Reality trace. (b) UCSD trace.

v_2 and v_3 during the warm-up period is 60 seconds, it is claimed to be 30 seconds during the attack scenario. As shown in Figure 2(c), HRP replicates more packets towards v_2 and v_3 during the attack scenario. The total number of hijacked packets by v_2 and v_3 is decreased while increasing the on-proportion because HRP decreases the replication factor when the network becomes more connected [10] [53]. For Prophet, the total number of routed packets via v_2 and v_3 increases for the attack scenario compared to the normal scenario (Figure 2(d)).

We repeated the same experiments above when v_2, v_3, and v_6 are compromised. During the attack scenario, the ICTs among (v_2-v_3) and (v_3-v_6) are claimed to be 0.5 of the corresponding ICTs during the normal scenario. As shown in Figures 2(e) and 2(f), the total number of hijacked packets by v_2, v_3, v_6 are higher compared to the experiments when only v_2 and v_3 are compromised for HRP and Prophet. We also repeated the experiments when v_2, v_3, v_6, and v_9 are compromised (they fake the ICTs of (v_2-v_3), (v_3-v_6), and (v_3-v_9) to be 0.5 of the corresponding ICTs during the normal scenario). Similar to the above conclusion, when Eve compromises more nodes, she is able hijack more packets. The total number of hijacked packets in Figures 2(g) and 2(h) increases compared to the total number of hijacked packets in Figures 2(e) and 2(f), respectively.

3.2 CollusiveHijack in the ONE simulator

We conducted trace-driven simulations in the ONE simulator [63] using the Reality [15] and the UCSD [16] real-world mobility traces. For the Reality trace, which consists of 97 users from MIT students, faculty and staff members with Bluetooth connection events over nine months, we set the duration of each experiment to 5 weeks including a warm-up period for 1 week. The 97 nodes run Prophet protocol [12] with $\beta = 0.9$ and $\gamma = 0.999$. After the warm-up period (during the second week), we randomly generated 100 messages, each with size = 1KB and deadline = 10 days, between the nodes with id $\in$ [1, 50]. For each experiment, we considered two scenarios, a normal scenario and an attack scenario. During the normal scenario, the nodes do not fake their ICTs. However, during the attack scenario, we randomly choose nodes from the set of nodes with id $\in$ [51, 97] and fake their ICTs during the warm-up period. Then, we compared the total number of routed packets during the normal and attack scenarios while varying the number of compromised nodes from 2 to 4 and the *lying factor* from 2 to 8. The

lying factor reflects the ratio that the compromised nodes fake their ICTs by (i.e., $ICT_{faked} = \frac{ICT_{original}}{lying\ factor}$). We repeated each experiment 100 times and we averaged these 100 runs per each experiment. Figure 3(a) shows the total number of routed packets for two, three, and four compromised nodes compared with the normal scenarios (i.e., when the nodes are not compromised). As shown in this figure, the total number of hijacked packets increases while increasing the number of the compromised nodes and their lying factor.

For the UCSD [16] trace, which consists of 275 PDA users from University of California - San Diego with Wi-Fi connection events over eleven weeks, after the first week of warm-up period, we randomly generated 100 messages, each with size = 1KB and deadline = 10 days, between the nodes with id $\in$ [1, 150]. For each experiment, during the normal scenario, the nodes do not fake their ICTs. However, during the attack scenario, we randomly choose nodes from the set of nodes with id $\in$ [151, 275] and fake their ICTs during the warm-up period. Then, we compared the total number of routed packets during the normal and attack scenarios while varying the number of compromised nodes from 2 to 4 and the *lying factor* from 2 to 8. Similar to the conclusions of the Reality trace's experiments above, for the UCSD trace, the total number of hijacked packets increases when the attacker compromises more nodes and when the lying factor increases, as shown in Figure 3(b).

4 COLLUSIVEHIJACK DETECTION

In this section, we present the KS2ST and the design of PDT, HDT, and EHDT.

4.1 The Kolmogorov-Smirnov two-sample test (KS2ST)

We represent the OMN by an undirected graph $G = (V, E)$, where V is a set of nodes and E is a set of links, as shown in Figure 4. The link between any two nodes, v_i and v_j in V, is denoted by $e_{i,j}$. If $1/\lambda_{i,j}$ is the ICT between v_i and v_j, then, the link weight of $e_{i,j}$ is defined as the contact frequency (i.e., $\lambda_{i,j}$), as shown in Figure 4. If v_i never meets v_j, then $e_{i,j}$ will not be in E. Accordingly, any two connected nodes in G should have at least one contact. Message forwarding from v_i to v_j can be accomplished during the contact event. If $D_{i,j}$, $ICT_{i,j}$ represent the link delay and the ICT between two uncorrelated nodes v_i and v_j in V, respectively, then:

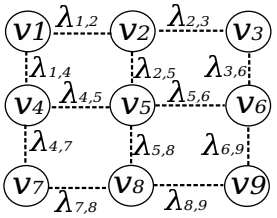


Fig. 4: OMN contact graph

$P[D_{i,j} \leq d] = \frac{1}{\mathbb{E}[ICT_{i,j}]} \int_0^d (1 - P[ICT_{i,j} \leq z])dz$ and

$\mathbb{E}[D_{i,j}] = \frac{\mathbb{E}[ICT_{i,j}]}{2} + \frac{\sigma^2(ICT_{i,j})}{2\mathbb{E}[ICT_{i,j}]}$ [13], where $\sigma^2(ICT_{i,j})$ is the variance of $ICT_{i,j}$.

Previous research [10] [13] [14] show that the probability density function (pdf) of the $ICT_{i,j}$ between v_i and v_j follows the exponential distribution (i.e., $\lambda_{i,j}e^{-\lambda_{i,j}t}$). We also validated that, using the Reality [15] and the UCSD [16] traces, and found that the pdfs of all ICTs among the users in these traces follow the exponential distribution. Accordingly, we can assume that $\mathbb{E}[D_{i,j}] = \mathbb{E}[ICT_{i,j}]$.

If we assume that $P_{v_i,v_j} = \{v_i, v_1, v_2, .., v_{\eta-1}, v_j\}$ represents a path between v_i and v_j in G, where v_x is the x_{th} relay node and η is the number of hops. Then, the path delay consists of links delays that are independently and exponentially distributed and pdf of P_{v_i,v_j} delay:

$$= \lambda_{i,1}e^{-\lambda_{i,1}t} + \lambda_{1,2}e^{-\lambda_{1,2}t} + .. + \lambda_{\eta-1,j}e^{-\lambda_{\eta-1,j}t}$$

, which follows

$$\sim Hypo(\lambda_{i,1}, \lambda_{1,2}, .., \lambda_{\eta-1,j})$$

with mean = $1/[\lambda_{i,1} + \lambda_{1,2} + .. + \lambda_{\eta-1,j}]$ and variance = $1/[\lambda_{i,1}^2 + \lambda_{1,2}^2 + .. + \lambda_{\eta-1,j}^2]$ [64].

The KS2ST is a well known non-parametric goodness-of-fit test [46]. This test measures the distance between the empirical cumulative distribution functions (CDFs) of two samples $\mathbf{a} = \{a\}_{i=1}^n$ and $\mathbf{b} = \{b\}_{i=1}^m$ to determine whether they have been drawn from the same distribution or not. The KS test, $KS.test(a, b)$, for $\{a\}_{i=1}^n$ and $\{b\}_{i=1}^m$ samples is defined as:

$D_{a,b} = \sup_{x \in a \cup b} |F_a(x) - F_b(x)|$, where sup is the supremum function and F_a, F_b are the empirical CDFs.

$F_a(x) = \frac{1}{n}\sum_{i=1}^n I_{\{a_i \leq x\}}$ and $F_b(x) = \frac{1}{m}\sum_{i=1}^m I_{\{b_i \leq x\}}$, where $I_{\{a_i \leq x\}}$ is the indicator function that has the value 1 if $a_i \leq x$, and 0 otherwise (same for $I_{\{b_i \leq x\}}$).

The null hypothesis of the two samples KS test (i.e., the samples are drawn from the same distribution) is rejected at significance level of $\alpha \in (0, 1]$ if:

$D_{a,b} > c(\alpha) \times \sqrt{\frac{n+m}{nm}}$, where $c(\alpha) = \sqrt{-\frac{1}{2}\ln(\frac{\alpha}{2})}$ and n, m are the sizes of **a** and **b** samples, respectively.

In our design we use $\alpha = 0.05$. Accordingly, we reject the null hypothesis if the $p - value$ [65] of the KS2ST is < 0.05.

4.2 Path Detection Technique (PDT)

Before presenting the details of PDT, we present the steps at each node when it routes a packet, as shown in Algorithm 1. *Note: the steps shown in oval-boxes are for HDT and EHDT that we present in Sections 4.3 and 4.4, respectively*. When a packet is available at the sender's buffer, the sender inserts the packet's creation time (PCT) and $Sender\_Id$ into the packet header. The sender also signs the inserted PCT and $Sender\_Id$ (lines 2-3 of Algorithm 1). Then, the sender waits until the next hop is available to forward the packet (lines 4 and 7 of Algorithm 1). When any node receives a packet to be routed, it verifies the signed Id of the previous hop using its public key (line 10 of Algorithm 1). When the next hop is available, the carrier node inserts its $Carrier\_Id$ into the packet header and signs it before forwarding the packet (lines 12 and 15 of Algorithm 1). The aforementioned steps can be done by including the nodes' payload integrity blocks in the packets (pages 24-25 of BSP [51]).

PDT is a detection-based protocol that runs by the destination nodes in OMNs. The pseudo code of PDT is presented in Algorithm 2. When the destination receives a

Algorithm 1 Steps at each node for PDT, HDT, and EHDT

1: **if** Sender **then**
2: Packet_Creation_Time (PCT) = current_time()
3: Insert&Sign$_{PrivK}$(PCT, $Sender\_Id$)→ Packet_Header (PH)
4: **Wait until** Next hop is available
5: (Packet_Receiving_Time (PRT) = current_time())
6: (Insert&Sign$_{PrivK}$(PRT) → Packet_Header (PH))
7: Forward the Packet
8: **else if** Carrier **then**
9: **if** Packet received **then**
10: Verify$_{PrevHop_{PubK}}$($PrevHop\_Id$, (PRT))
11: **Wait until** Next hop is available
12: Insert&Sign$_{PrivK}$($Carrier\_Id$) → Packet_Header (PH)
13: (Packet_Receiving_Time (PRT) = current_time())
14: (Insert&Sign$_{PrivK}$(PRT) → Packet_Header (PH))
15: Forward the Packet

Algorithm 2 PDT at the destination nodes

1: **for each** received $Packet$ via a $Path$ with η hops **do**
2: Packet_delay (Pd) = current_time() - PCT
3: $Path$ = get_path(PH) = $\{v_i, v_1, v_2, .., v_{\eta-1}, v_j\}$
4: **for** the $Sender$ and **each** $Carrier$ in $Path$ **do**
5: Verify$_{Sender_{PubK}}$($Sender\_Id$)
6: Verify$_{Carrier_{PubK}}$($Carrier\_Id$)
7: Save Pd for $Path$
8: **for k** received $Packets$ via a $Path$ with η hops **do**
9: $< Pd_1, Pd_2, .., Pd_k >$ = get_packet_delays($Path$)
10: $< \lambda_1, \lambda_2, .., \lambda_\eta >$ = get_λ's($Path$)
11: $reject_{null\_hypoehsis}$ = 0
12: **for** j = 1 to $num_{tests} = 10000$ **do**
13: $< \hat{Pd_1}, \hat{Pd_2}, .., \hat{Pd_{10k}} >$ = $get_{rand}(Hypo(\lambda_1, \lambda_2, .., \lambda_\eta))$
14: $p\text{-}value = KS.test(< Pd_1, Pd_2, .., Pd_k >, < \hat{Pd_1}, \hat{Pd_2}, .., \hat{Pd_{10k}} >)$
15: **if** $p\text{-}value < 0.05$ **then**
16: $reject_{null\_hypoehsis} = reject_{null\_hypoehsis} + 1$
17: **if** $(reject_{null\_hypoehsis}/num_{tests}) > 0.05$ **then**
18: $Path$ is compromised
19: **else**
20: $Path$ is not-compromised

packet, it calculates the packet's delay and finds its path. The destination verifies the inserted Ids into the packet header using the sender and carrier nodes' public keys (lines 2-6 of Algorithm 2). For all received packets via the same path, e.g., $P_{v_i,v_j} = \{v_i, v_1, v_2, .., v_{\eta-1}, v_j\}$, the destination v_j, stores the packets' delays (line 7 of Algorithm 2) and finds the contact frequencies of P_{v_i,v_j} (i.e., $\lambda_{i,1}, \lambda_{1,2}, .., \lambda_{\eta-1,j}$). These contact frequencies are announced to all nodes in the OMN and used by HRP and Prophet to make replication decisions. v_j employs the KS2ST to determine whether the delays of the received packets match the summation of the announced/claimed links' delays of P_{v_i,v_j} (that follows $\sim Hypo(\lambda_{i,1}, \lambda_{1,2}, .., \lambda_{\eta-1,j})$. In the following paragraph, we present the steps of PDT through an example.

We assume that v_1 in Figure 4 sent k packets with delays = $< Pd_1, Pd_2, .., Pd_k >$ to v_9 via $P_{v_1,v_9} = \{v_1, v_2, v_3, v_6, v_9\}$ path that has the following contact frequencies: $< \lambda_{1,2}, \lambda_{2,3}, \lambda_{3,6}, \lambda_{6,9} >$. Once v_9 collects the packets' delays and contact frequencies (lines 9-10 in Algorithm 2), it repeats the following process 10,000 times. v_9 draws a random sample, with size = $10\times k$, $< \hat{Pd_1}, \hat{Pd_2}, .., \hat{Pd_{10k}} >$ from $Hypo(\lambda_{1,2}, \lambda_{2,3}, \lambda_{3,6}, \lambda_{6,9})$, and performs a KS2ST between the received packets delays and the drawn sample (lines 13-14 of Algorithm 2). If the resulted $p\text{-}value$ of the KS2ST is less than α = 0.05, v_9 increases the $reject_{null\_hypoehsis}$ counter. This counter keeps track of the number of times the null hypothesis is rejected. After repeating the KS2ST 10,000 times (with different random samples), if the ratio of the number of times that the null hypothesis is rejected is $> 5\%$, then v_9 labels P_{v_1,v_9} as a compromised path. Otherwise, P_{v_1,v_9} is labeled as a not-compromised path (lines 17-20 of Algorithm 2).

4.3 Hop Detection Technique (HDT)

HDT is a detection-based protocol that runs by the destination nodes in OMNs. HDT is based on the idea that collecting the packets' receiving times (PRTs) at the intermediate nodes enhances the destination node detection capability against the CollusiveHijack attack. That is, instead of labeling the whole path as compromised, as the case of PDT, HDT aims to pinpoint the lying hops and accordingly the compromised nodes in the OMN. However, HDT requires additional steps to be performed by the intermediate nodes apart from the steps accomplished by BSP [51], as shown in oval-boxes of Algorithm 1. When the next hop is available for the sender or the intermediate nodes, they have to insert the PRT into the packet header and sign the inserted PRT before delivering the packet to the next hop, as shown in lines 5-6 and 13-14 of Algorithm 1. Moreover, when an intermediate node, e.g., v_i, receives a packet, it verifies whether the PRT (that has been signed by the previous hop) matches its time. Also, v_i verifies the signature of the previous hop using the $PrevHop_{PubK}$ (line 10 of Algorithm 1). Notice that if we ignore the links' propagation and transmission delays (relatively small compared to ICTs in OMNs), the signed time by the previous hop should equal the PRT at v_i.

The pseudo code of HDT is shown in Algorithm 3. To illustrate how HDT works, we present its steps using the same example we used in Section 4.2 (node v_1, in Figure 4, sent k packets to node v_9 via $P_{v_1,v_9} = \{v_1, v_2, v_3, v_6, v_9\}$). Once v_9 receives a packet from v_6, it verifies the PRT and the Id of v_6 using v_6's public key, as shown in line 2 of Algorithm 3. Afterwards, v_9 gets the path of the received packet and verifies the Ids, PCT, and PRT's of v_1, v_2, v_3, and v_6 using their public keys (lines 3-6 of Algorithm 3).

Once v_9 collects the packets' delays for all intermediate hops, it builds the packet receiving times matrix, PRT_{nodes}, as shown in line 8 of Algorithm 3. PRT_{nodes} is a $[k \times (\eta+1)]$ matrix that contains the creation and receiving times for the k packets that have been routed via a path with η hops. The $(k \times 5)$ PRT_{nodes} that is built by v_9 is shown in Figure 5. Notice that $PCT_{x,y}$ represents the creation time of the x_{th} packet at node y and $PRT_{x,y}$ represents the receiving time of the x_{th} packet at node y. By subtracting each i_{th} column from the $i_{th}+1$ column in PRT_{nodes}, v_9 builds the D_{hops}, an $(k \times \eta) = (k \times 4)$ matrix that contains the hops' delays of P_{v_1,v_9} (lines 9-11 of Algorithm 3). Notice that $D_{x,(y,w)}$

Algorithm 3 HDT at the destination nodes

1: **for each** received $Packet$ via a $Path$ with η hops **do**
2: Verify$_{PrevHop_{PubK}}(PrevHop_Id, PRT)$
3: $Path$ = get_path(PH) = $\{v_i, v_1, v_2, .., v_{\eta-1}, v_j\}$
4: **for** the $Sender$ and **each** $Carrier$ in $Path$ **do**
5: Verify$_{Sender_{PubK}}(Sender_Id, PCT_{Sender_Id})$
6: Verify$_{Carrier_{PubK}}(Carrier_Id, PRT_{Carrier_Id})$
7: **for k** received $Packets$ via a $Path$ with η hops **do**
8: $PRT_{nodes} \leftarrow$ $\begin{bmatrix} PCT_{1,v_i} & PRT_{1,v_1} & .. & PRT_{1,v_j} \\ PCT_{2,v_i} & PRT_{2,v_1} & .. & PRT_{1,v_j} \\ .. & .. & .. & .. \\ PCT_{k,v_i} & PRT_{k,v_1} & .. & PRT_{1,v_j} \end{bmatrix}_{\eta+1}^{k}$
9: **for each** col_i in PRT_{nodes} **do**
10: $D_{hops}[col_i] = PRT_{nodes}[col_{i+1}]$ - $PRT_{nodes}[col_i]$
11: $D_{hops} = \begin{bmatrix} D_{1,(v_i,v_1)} & D_{1,(v_1,v_2)} & .. & D_{1,(v_{\eta-1},v_j)} \\ D_{2,(v_i,v_1)} & D_{2,(v_1,v_2)} & .. & D_{2,(v_{\eta-1},v_j)} \\ .. & .. & .. & .. \\ D_{k,(v_i,v_1)} & D_{k,(v_1,v_2)} & .. & D_{k,(v_{\eta-1},v_j)} \end{bmatrix}_{\eta}^{k}$
12: **for each** col_i in D_{hops} **do**
13: $< D_1, D_2, .., D_k >$ = transpose(col_i)
14: $< \lambda_i >$ = get_$\lambda(Link_i)$
15: $reject_{null\_hypoehsis}$ = 0
16: **for** j = 1 to $num_{tests} = 10000$ **do**
17: $< \hat{D_1}, \hat{D_2}, .., \hat{D_{10k}} > = get_{rand}(Hypo(\lambda_i))$
18: $p\text{-}value = KS.test(< D_1, D_2, .., D_k >, < \hat{D_1}, \hat{D_2}, .., \hat{D_{10k}} >)$
19: **if** $p\text{-}value < 0.05$ **then**
20: $reject_{null\_hypoehsis} = reject_{null\_hypoehsis} + 1$
21: **if** $(reject_{null\_hypoehsis}/num_{tests}) > 0.05$ **then**
22: Source Node in Hop i is compromised
23: **else**
24: Hop i is not-compromised

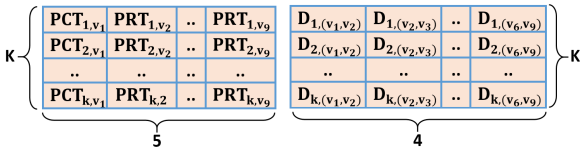


Fig. 5: PRT_{nodes} and D_{hops} built by v_9

represents the delay of the x_{th} packet at $(y) \longleftrightarrow (w)$ hop. The $(k \times 4)$ D_{hops} that is built by v_9 is shown in Figure 5. Then, v_9 performs the same detection steps as the PDT algorithm by leveraging the KS2ST. However, since the hops' delays are calculated by v_9 (i.e., the columns of D_{hops}), HDT performs a hop-wise detection for P_{v_1,v_9}. Consequently, v_9 can label each of the source nodes in the intermediate hops, $[(v_1, v_2), (v_2, v_3), (v_3, v_6), (v_6, v_9)]$, as a compromised or not-compromised node (lines 12-24 of Algorithm 3).

4.4 Early Hop Detection Technique (EHDT)

EHDT aims to early detect the CollusiveHijack attack at the intermediate nodes that carry the packets instead of delaying the detection until the packets are received by the destination nodes (as is the case of HDT). EHDT requires the additional steps that are shown in oval-boxes of Algorithm 1, in which the sender and the intermediate nodes insert and sign the PRTs before delivering the packets to the next hop. The intermediate nodes leverage the aforementioned PRTs of all packets that are routed across them to early detect the compromised nodes. The pseudo code of EHDT is presented in Algorithm 4. In order to illustrate how EHDT works, let's assume that in addition to the example that we presented in Section 4.3 in which v_1, in Figure 4, sent k packets to v_9 via $P_{v_1,v_9} = \{v_1, v_2, v_3, v_6, v_9\}$, also, v_4 sent m packets to v_6 via $P_{v_4,v_6} = \{v_4, v_1, v_2, v_3, v_6\}$. In the following paragraph, we present how EHDT works on v_3.

Once v_3 receives a packet from v_2, it verifies the PRT and the Id of v_2 using v_2's public key, as shown in line 2 of Algorithm 4. Afterwards, v_3 gets the last hop of the received packet and verifies the Ids, PCT, and PRT's of v_1 and v_2 using their public keys (lines 3-5 of Algorithm 4). Notice that for v_z in the $(v_x) \longleftrightarrow (v_y) \longleftrightarrow (v_z)$ path, $get\_last\_hop(PH)$ function returns the headers of the packets that have been routed via $(v_x) \longleftrightarrow (v_y)$ link. In line 4 of Algorithm 4, v_z verifies either the PCT or PRT of v_x based on whether v_x is the source or the carrier node of the packet, respectively. In our example, v_3 verifies the PCT's of the k packets that have been sent from v_1 to v_9 via P_{v_1,v_9} and the PRT's of the m packets that have been sent from v_4 to v_6 via P_{v_4,v_6}. In the next step, v_3 collects the $k + m$ packets' delays for the $(v_1) \longleftrightarrow (v_2)$ hop and builds the packet receiving times matrix, PRT_{nodes}, as shown in lines 6-7 of Algorithm 4. PRT_{nodes} is a $[(k + m) \times (2)]$ matrix that contains the creation and receiving times for the k and m packets, respectively, that have been routed via $\{v_1, v_2\}$ link. By subtracting the first column from the second column in PRT_{nodes}, v_3 builds the D_{hops}, an $[(k + m) \times 1]$ vector that contains the link delay of $(v_1) \longleftrightarrow (v_2)$ (lines 8-9 of Algorithm 4). Then, v_3 performs the same detection steps as the HDT algorithm by leveraging the KS2ST and labels the source node, v_1, in the $(v_1) \longleftrightarrow (v_2)$ hop, as either a compromised or not-compromised node (lines 10-21 of Algorithm 4). Notice that $\{v_2, v_3\}$ link is also shared amongst P_{v_1,v_9} and P_{v_4,v_6} paths. Hence, the k and m packets that are delivered via $\{v_2, v_3\}$ can be leveraged by v_6 to detect the CollusiveHijack attack even though these packets belong to two different paths (i.e., this is not the case for HDT since the detection is only accomplished at the destination nodes). Accordingly, EHDT enhances the detection latency of CollusiveHijack attack, as we will illustrate in Section 6.

5 Implementation and Experimental Setup

We implemented PDT, HDT, and EHDT using R statistical language [66] on Asus Eee notebooks (shown in Figure 2(a)). The notebooks run Ubuntu 14.04 LTS and have Intel(R) Atom(TM) CPU operating at 1.6 GHz and 1GB RAM. Our R code is executed at the destination nodes for both PDT and HDT and at the intermediate nodes for EHDT by the C++ user-space daemon services of HRP and Prophet protocols [53] [54]. We leveraged the Rcpp package [67] for the seamless integration of R and C++. We enabled BSP [51] for HRP and Prophet daemon services. For the cryptographic sign and verify operations, we leveraged the Pairing-Based Cryptography (PBC) library [68] that implements the Hess identity-based signatures [69]. The PBC implementation uses

Algorithm 4 EHDT at the intermediate nodes

1: **for each** received $Packet$ via a $Link$ **do**
2: Verify$_{PrevHop_{PubK}}(PrevHop_Id, PRT)$
3: $Link$ = get_last_hop(PH) = $\{v_x, v_y\}$
4: Verify$_{v_{x\,PubK}}(v_x_Id, PCT_{v_x_Id}/PRT_{v_x_Id})$
5: Verify$_{v_{y\,PubK}}(v_y_Id, PRT_{v_y_Id})$
6: **for k+m** received $Packets$ via $Link$ **do**
7: $PRT_{nodes} \leftarrow$

PCT_{1,v_x}	PRT_{1,v_y}
PCT_{2,v_x}	PRT_{2,v_y}
PCT_{3,v_x}	PRT_{3,v_y}
..	..
PCT_{k,v_x}	PRT_{k,v_y}
PRT_{k+1,v_x}	PRT_{k+1,v_y}
PRT_{k+2,v_x}	PRT_{k+2,v_y}
PRT_{k+3,v_x}	PRT_{k+3,v_y}
..	..
PRT_{k+m,v_x}	PRT_{k+m,v_y}

$k+m$; 2

8: $D_{hops}[col_1] = PRT_{nodes}[col_2^2] - PRT_{nodes}[col_1]$
9: $D_{hops} =$

$D_{1,(v_x,v_y)}$
$D_{2,(v_x,v_y)}$
..
$D_{k+m,(v_x,v_y)}$

$k+m$; 1

10: $< D_1, D_2, .., D_k >$ = transpose(col_1)
11: $< \lambda_i >$ = get_$\lambda(Link)$
12: $reject_{null\_hypoehsis}$ = 0
13: **for** j = 1 to $num_{tests} = 10000$ **do**
14: $< \hat{D_1}, \hat{D_2}, .., \hat{D_{10(k+m)}} > = get_{rand}(Hypo(\lambda_i))$
15: $p\text{-}value = KS.test(< D_1, D_2, .., D_{k+m} >, < \hat{D_1}, \hat{D_2}, .., \hat{D_{10(k+m)}} >)$
16: **if** $p\text{-}value < 0.05$ **then**
17: $reject_{null\_hypoehsis} = reject_{null\_hypoehsis} + 1$
18: **if** $(reject_{null\_hypoehsis}/num_{tests}) > 0.05$ **then**
19: v_x in $\{v_x, v_y\}$ is compromised
20: **else**
21: $\{v_x, v_y\}$ is not-compromised

a 160-bit elliptic curve group with 512-bit keys. In the following paragraphs, we describe our experiments.

1) **Experiments in the ONE simulator:** we extracted the ICTs and the number of hops from the experiments that we conducted in Section 3.2 in the ONE simulator [63] using the Reality [15] and UCSD [16] mobility traces. We aim to investigate the performance of PDT, HDT and EHDT for various scenarios. Hence, we performed different experiments by varying the ratio of the compromised links in different paths (with 4, 5, and 6 hops). The ratio of the compromised links in our experiments are: (1) $16.6\%(\frac{1}{6})$. (2) $20\%(\frac{1}{5})$. (3) $25\%(\frac{1}{4})$. (4) $33.3\%(\frac{2}{6})$. (5) $40\%(\frac{2}{5})$. (6) $50\%(\frac{3}{6})$. (7) $60\%(\frac{3}{5})$. We also varied the *lying factor* (illustrated in Section 3.2) of the compromised links (from 2 to 8) and the number of the received packets, k, at the destination from 20 to 100.

2) **Experiments on Asus notebooks testbed:** we repeated the four experiments against HRP and Prophet on the testbed that are shown in Figure 2(a) for 0.4 and 0.9 on-off ratios, respectively. We generated 100 data packets (each with a deadline = 300 seconds) from v_1 to v_{11} and we varied the number of compromised nodes from 2 to 4. The lying factor for these experiments is 2. We leveraged the HRP implementation [53] and Prophet implementation [54] to collect the ICTs among the nodes. The HRP implementation uses Optimized Link State Routing (OLSR) [70] for topology maintenance by invoking *olsrd* [71], an OLSR implementation, and fetches topology information from it. We used *hrptclient* [53] and IBR-DTN's *dtnsend* and *dtnrecv* [54] for the data flow for HRP and Prophet, respectively. We measured the packet's delay at the destination nodes for both PDT and HDT and at the intermediate nodes for EHDT.

We evaluated PDT, HDT, and EHDT in two aspects:

1) **Quantifying the overhead** of PDT, HDT, and EHDT using the following metrics:
   a) **Packet Size Overhead:** $\frac{additonal\ data\ size}{packet\ size}$, which measures the overhead of adding the node's Id, PCT, and PRT into the packets.
   b) **Execution Time Overhead**: the required time to execute the sign and verify operations.
2) **Evaluating the performance** of PDT, HDT and EHDT using the following metrics:
   a) **Detection Rate:** the ability of PDT, HDT and EHDT to detect the attempts of the CollusiveHijack attacker (true positive rate).
   b) **False Positive Rate:** the rate of claiming a legitimate path/link as a compromised one.
   c) **Execution Time (sec):** the execution time of KS2ST at either the destination nodes for PDT and HDT and at the intermediate nodes for EHDT.
   d) **Detection Latency (days or hours):** the required time to collect 20, 40, 60, 80, and 100 packets at the destination nodes for both PDT and HDT and at the intermediate nodes for EHDT.

6 Performance Evaluation

In this section, we present the overhead of PDT, HDT, and EHDT as well as their performance evaluations.

6.1 Quantifying the Overhead of PDT, HDT, and EHDT

In PDT, HDT and EHDT, the sender adds its Id and PCT (each is 4B) into the packet's header and each intermediate node adds its Id (4B). Also, in both of HDT and EHDT, each intermediate node adds its PRTs (4B). The signature field of the Hess identity-based scheme is 64B. Hence, the packet size overhead of PDT in bytes for the $P_{v_i,v_j} = \{v_i, v_1, v_2, .., v_{\eta-1}, v_j\}$ path is $68\eta + 4$. For HDT and EHDT, the packet size overhead for P_{v_i,v_j} is 72η. For a 10-hop path with an IPv4 packet size (i.e., 64KB), the packet size overhead for PDT is $\sim$1.04% and $\sim$1.09% for both of HDT and EHDT, respectively, which are relatively small. In order to calculate the execution time of the sign and verify operations of the Hess identity-based scheme [69], we averaged 10,000 measurements of these operations on one Asus Eee notebooks and found that the execution time of the sign and verify operations are 118 msec and 42 msec, respectively.

6.2 Performance Evaluation of PDT, HDT, and EHDT

Evaluation in the ONE simulator using Reality trace. We conducted the first experiment on a path with η = 6 hops including one compromised hop. We performed the following

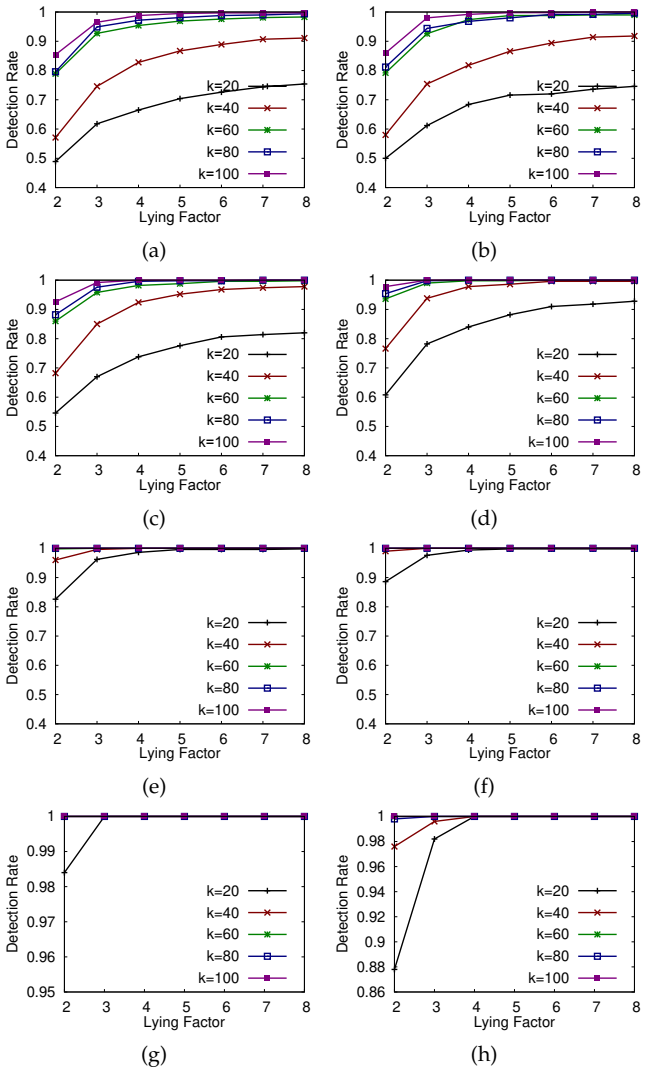


Fig. 6: PDT's Detection Rate for Reality trace when the ratio of the compromised links: (a) 16.6%($\frac{1}{6}$). (b) 20%($\frac{1}{5}$). (c) 25%($\frac{1}{4}$). (d) 33%($\frac{2}{6}$). (e) 40%($\frac{2}{5}$). (f) 50%($\frac{3}{6}$). (g) 60%($\frac{3}{5}$). (h) HDT and EHDT's Detection Rate.

steps. First, we extracted 1,000 different paths with 6 hops from the trace-driven 100 experiments that we conducted in Section 3.2. Second, we extracted the ICTs among the nodes in these paths and collected the packets' delays at the destination nodes. Third, we randomly picked one hop and varied its lying factor from 2 to 8. Fourth, we counted the number of times the destination, which runs the PDT algorithm, was able to detect the CollusiveHijack attack after receiving 20, 40, 60, 80, and 100 packets. Fifth, we averaged the 1,000 runs for each lying factor and k value.

The PDT's detection rate for the first experiment is shown in Figure 6(a). The detection rate increases when the two compromised nodes increase their lying factor and when the total number of the received packets at the destination increases. We illustrated in Figures 3(a) and 3(b) that it is attractive for the compromised nodes to increase their lying factor to hijack more packets. However, increasing the number of hijacked packets enhances the PDT's detection

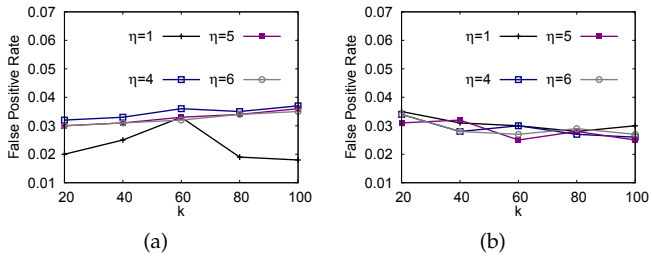


Fig. 7: False Positive Rate for (a) Reality trace of PDT (η = 4, 5, 6 hops) and of HDT and EHDT (η = 1 hop). (b) UCSD trace of PDT (η = 4, 5, 6 hops) and of HDT and EHDT (η = 1 hop).

capability. The PDT's detection rates for the first experiment when the destination receives 100 packets are: 85.4%, 96.5%, 98.8%, 99.5%, 99.7%, 99.7%, 99.8% for 2, 3, 4, 5, 6, 7, 8 lying factors, respectively, as shown in Figure 6(a).

We repeated the five steps mentioned above for a higher ratio of compromised links in the path. In the second experiment, we had a path with η = 5 hops including one compromised hop. As it is clear in Figure 6(b), we can draw the same conclusions as from Figure 6(a). The PDT's detection rate increases when the compromised nodes increase their lying factor and when the destination receives more packets. The aforementioned conclusions can be also observed for the remaining five experiments that are shown in Figures 6(c)-6(g). Moreover, if we compare the PDT's detection rate of all figures, we observe that while Eve increases the ratio of the compromised links, the PDT's detection rate increases. Figure 7(a) presents the false positive rate for PDT when η = 4, 5, and 6 hops. The false positive rate for all experiments is $< 3.6\%$, which is relatively small.

Since HDT and EHDT perform a hop-wise detection against the CollusiveHijack attack, they have the same detection rate, which is presented for 1-hop for different k values in Figure 6(h). As it is clear in Figure 6(h), HDT and EHDT's detection rates are $> 98\%$ for all lying factors when $k > 40$. Hence, we recommend to leverage either HDT or EHDT to effectively detect the CollusiveHijack attack despite it's overhead and incompatibility with BSP [51]. Moreover, HDT and EHDT have the same false positive rates (i.e., on 1-hop), which are relatively small (i.e., $< 3.0\%$), as we presented in Figure 7(a) (i.e., when η=1).

The execution time of PDT is presented in Figure 8(a). PDT's execution time increases linearly and remains relatively short (< 2.2 seconds) while increasing k and η (i.e., the total number of received packets and the number of hops, respectively). For HDT, the execution time at the destination node when η = 1, 2, 3, 4, 5, or 6 increases linearly from 1.2 to 11.5 seconds while increasing k, as presented in Figure 8(b). The execution time when η = 1 in Figure 8(b) also represents EHDT's execution time since EHDT is a 1-hop detection algorithm that is run by the intermediate nodes. That is, EHDT's execution time = HDT's execution time when η = 1.

For the detection latency, we calculated the average time needed to receive 20, 40, 60, 80, and 100 packets for PDT, HDT, and EHDT using the Reality trace. Since the detection of the CollusiveHijack attack for both PDT and HDT is

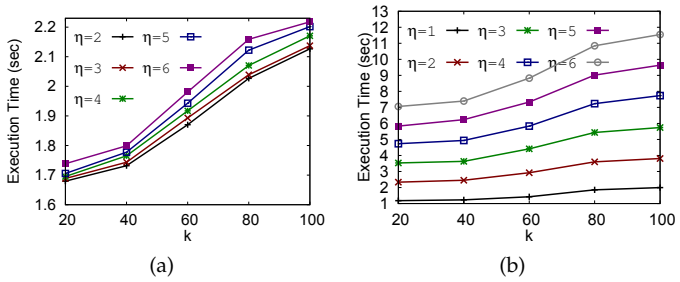


Fig. 8: Execution Time of (a) PDT. (b) HDT when η = 1, 2, 3, 4, 5, or 6 and EHDT when η = 1.

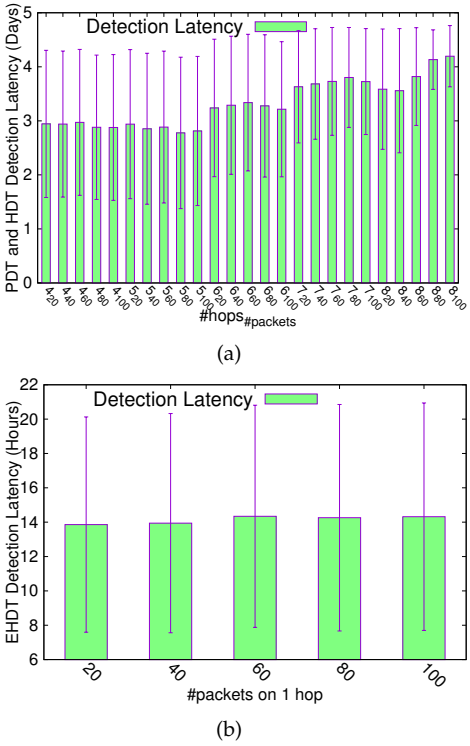


Fig. 9: Detection Latency for different number of hops and packets using Reality trace for (a) PDT and HDT. (b) EHDT.

accomplished at the destination nodes, they both incur the same detection latency value for the same number of packets and hops. As presented in Figure 9(a), the detection latencies for different number of packets (i.e., 20, 40, 60, 80, and 100 packets) for different number of hops (i.e., 4, 5, 6, 7, and 8 hops) for both PDT and HDT varies between 3 to 4 days, which are relatively acceptable in OMNs. However, for EHDT, since the intermediate nodes in OMNs perform a hop-wise detection, the detection latencies for 20, 40, 60, 80, and 100 packets on 1-hop are ∼14 hours, as presented in Figure 9(b). That is, EHDT enhances the detection latencies ∼85% compared to PDT and HDT and enables the nodes to detect the CollusiveHijack attack faster than the case of PDT and HDT. That's why we recommend to leverage EHDT to early detect the CollusiveHijack attack in OMNs.

Evaluation in the ONE simulator using UCSD trace. We calculated the PDT, HDT, and EHDT's detection rates for UCSD trace with the same ratio of compromised hops,

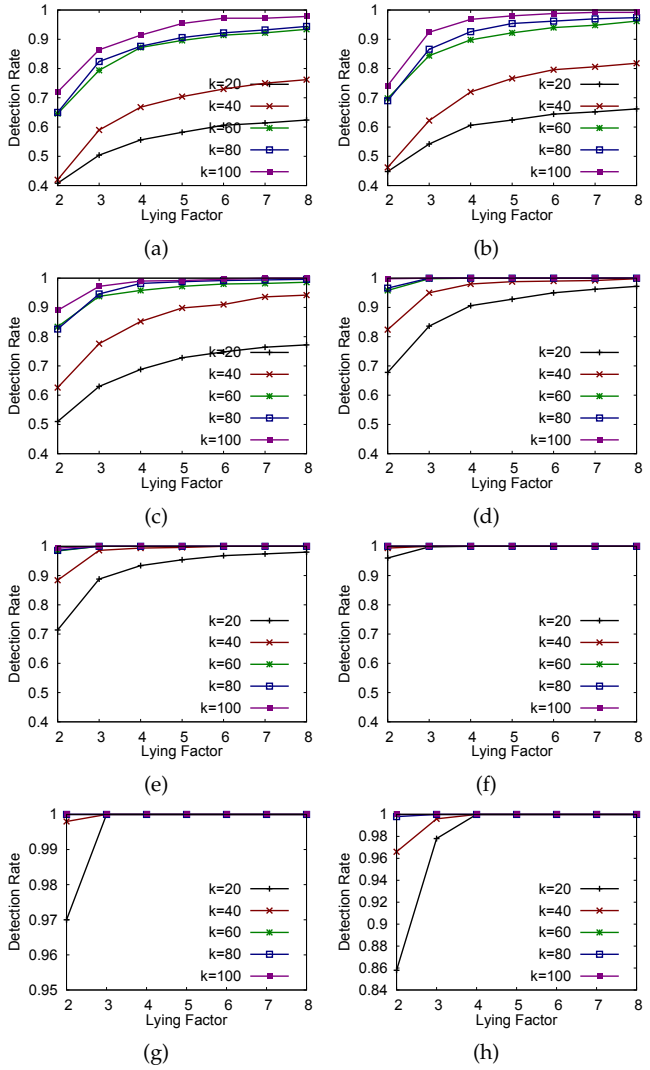


Fig. 10: PDT's Detection Rate for UCSD trace when the ratio of the compromised links: (a) 16.6%($\frac{1}{6}$). (b) 20%($\frac{1}{5}$). (c) 25%($\frac{1}{4}$). (d) 33%($\frac{2}{6}$). (e) 40%($\frac{2}{5}$). (f) 50%($\frac{3}{6}$). (g) 60%($\frac{3}{5}$). (h) HDT and EHDT's Detection Rate.

lying factors, and received packets at the destination node, as we did for the Reality trace above. The detection rates for PDT are presented in Figures 10(a), 10(b), 10(c), 10(d), 10(e), 10(f), and 10(g). The detection rates for HDT and EHDT are presented in 10(h). We obtain the same conclusions from the aforementioned figures as from the experiments with the Reality trace above. That is, the PDT's detection rate increases when Eve increases the ratio of the compromised links and/or the lying factors, as well as when the destination receives more packets. Similarly, HDT and EHDT's detection rates increase when Eve increases the lying factors and when the destination node receives more packets. We also calculated the false positive rates for PDT, HDT, and EHDT using UCSD trace and found that their false positive rates are relatively small, as presented in Figure 7(b).

We also calculated the detection latencies for PDT, HDT, and EHDT using the UCSD trace. Similar to the conclusions that we derived using the Reality trace above, EHDT

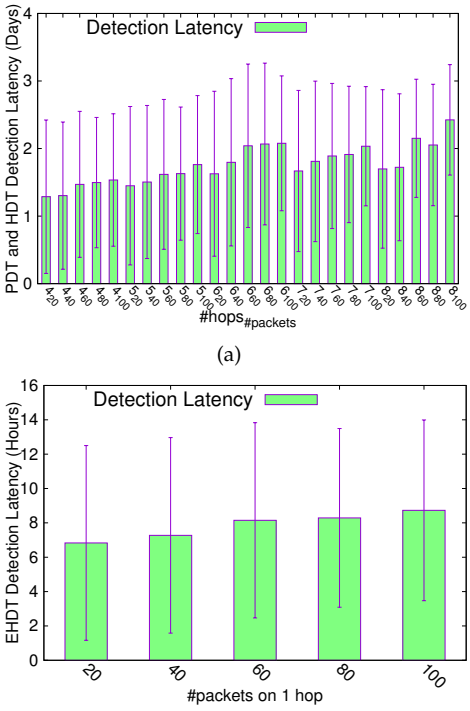


Fig. 11: Detection Latency for different number of hops and packets using UCSD trace for (a) PDT and HDT. (b) EHDT.

enhances the detection latencies of CollusiveHijack attack compared to PDT and HDT. That is, the detection latencies for PDT and HDT for 20, 40, 60, 80, and 100 packets for 4, 5, 6, 7, and 8 hops are 1.2 to 2.6 days, which are reduced to 7 to 9 hours for the case of EHDT (i.e., $\sim$ 75% to 85% enhancement), as shown in Figures 11(a) and 11(b), respectively.

System evaluation on Asus notebooks testbed. Figures 12(a) and 12(b) show the detection rate of PDT for the HRP experiments that we conducted on the testbed in Figure 2(a) using 0.4 and 0.9 on-off ratios. As is clear in these figures, the detection rate increases when the total number of received packets at v_{11} increases and when the number of compromised nodes increases from 2 to 4. When v_{11} receives 60 packets, the PDT detection rate is $>$ 88.0%. We also derived the same conclusions for the detection rate of PDT for the Prophet experiments on the testbed, as shown in Figures 12(c) and 12(d). For both HDT and EHDT, the detection rate against the CollusiveHijack attack for these experiments for both HRP and Prophet is always $>$ 98.0% when $k > 40$. Moreover, the false positive rates of PDT, HDT, and EHDT decreases when increasing the number of compromised nodes and the number of received packets, k, at the destination nodes, as presented in Figures 13(a) and 13(b) for the HRP and Prophet experiments, respectively.

7 Security Analysis

In this section, we provide a security analysis and discuss different security attacks that might be launched by Eve.

1) Fake Ids attack against PDT. Eve might pretend that two or more of her nodes have the same

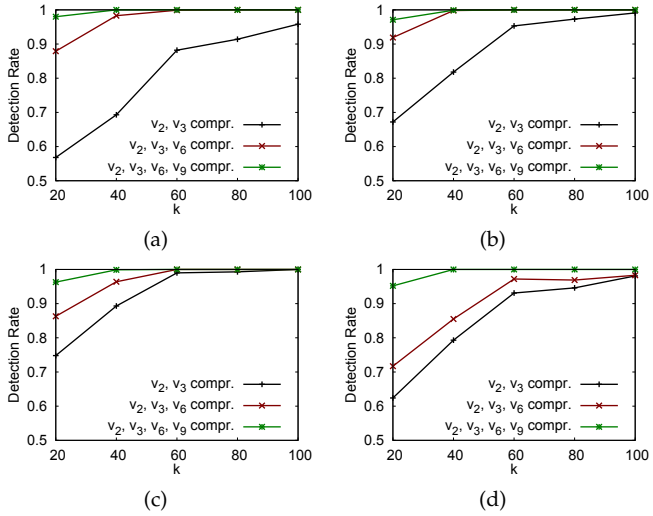


Fig. 12: PDT's Detection Rate on Asus notebooks testbed: for HRP using (a) 0.4 and (b) 0.9 on-off ratios and for Prophet using (c) 0.4 and (d) 0.9 on-off ratios.

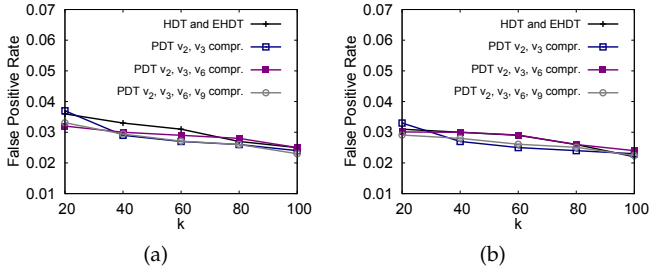


Fig. 13: PDT, HDT, and EHDT 's False Positive Rate on Asus notebooks testbed for (a) HRP and (b) Prophet, using 0.4 and 0.9 on-off ratios.

Id to not be detected by PDT. Without loss of generality, if we assume that v_2 and v_3 are compromised in $P_{s,d} = (s) \xleftrightarrow{\lambda_{s,v_1}} (v_1) \xleftrightarrow{\lambda_{v_1,v_2}} (v_2) \xleftrightarrow{\lambda_{v_2,v_3}} (v_3) \xleftrightarrow{\lambda_{v_3,v_4}} (v_4) \xleftrightarrow{\lambda_{v_4,d}} (d)$, then v_3 pretends to be v_2 (i.e., Eve aims to hide v_3 identity). In this case, the packets' delays at d follow $Hypo(\lambda_{s,v_1}, \lambda_{v_1,v_2}, \lambda_{v_2,v_3}, \lambda_{v_3,v_4}, \lambda_{v_4,d})$. However, when d runs PDT, the random samples (line 13 of Algorithm 2) are drawn from $Hypo(\lambda_{s,v_1}, \lambda_{v_1,v_2}, \lambda_{v_2,v_4}, \lambda_{v_4,d})$. In order to check whether PDT is able to detect fake Ids attacks, we repeated the first and second steps of the experiments in the ONE simulator (in Section 6.2) for η = 5, 6, and 7 hops for both Reality and UCSD traces. For the third step, we randomly picked two adjacent nodes and launched a fake Ids attack by pretending they have the same Id. Then, we averaged the number of times that the destination, which runs PDT, was able to detect the fake Ids attack for different number of received packets, k. As presented in Figures 14(a) and 14(b), the PDT detection rate is $>$ 96.0% when $k > 60$ for both Reality and UCSD traces. Hence, if Eve launches the fake Ids attack, one of her nodes will be detected (v_2 in $P_{s,d}$ example above). We also calculated the PDT's detection rates against fake Ids attack for 3 compromised nodes (all have same Id) and found that it is $>$ 99.0% when $k > 40$.

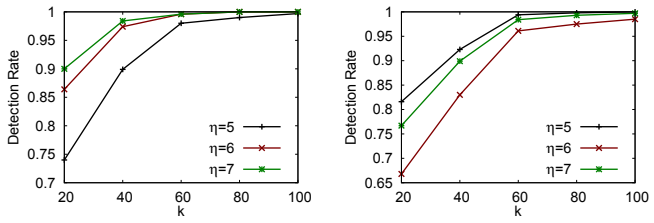


Fig. 14: (a) PDT Detection Rate Against Fake Ids Attack for a) Reality trace. b) UCSD trace

2) Fake PRTs attack against HDT and EHDT. Eve's nodes might fake their PRTs in order to not be detected by HDT and EHDT. For example, in $P_{s,d} = (s) \xleftrightarrow[\tau_1]{\lambda_{s,v_1}} (v_1) \xleftrightarrow[\tau_2]{\lambda_{v_1,v_2}} (v_2) \xleftrightarrow[\tau_3]{\lambda_{v_2,v_3}} (v_3) \xleftrightarrow[\tau_4]{\lambda_{v_3,v_4}} (v_4) \xleftrightarrow[\tau_5]{\lambda_{v_4,d}} (d)$ path, where τ's are the links delays, if s sends a packet at τ_0, the PCT and PRTs for this packet at $P_{s,d}$ path = $[\tau_0, \sum_{i=0}^{1}\tau_i, \sum_{i=0}^{2}\tau_i, \sum_{i=0}^{3}\tau_i, \sum_{i=0}^{4}\tau_i, \sum_{i=0}^{5}\tau_i]$. The aforementioned PCT and PRTs are respectively signed by $[(s), (s,v_1), (v_1,v_2), (v_2,v_3), (v_3,v_4), (v_4,d)]$, as presented in lines 3, 6, and 14 of Algorithm 1.

Without loss of generality, if Eve compromises both of v_2 and v_3 and fakes their contact frequencies to $2 \times \lambda_{v_2,v_3}$, she is also able to fake the PRT at v_3 to $\frac{\tau_3}{2}$ in order to not be detected by HDT and EHDT on the $(v_2) \xleftrightarrow[\frac{\tau_3}{2}]{2\times\lambda_{v_2,v_3}} (v_3)$ hop. That is, Eve fakes the PRT at v_3 to match her claimed contact frequency, $2 \times \lambda_{v_2,v_3}$. However, in this case, the destination node, d, in case of HDT or v_4 in case of EHDT are able to detect the source node, v_3, of $(v_3) \xleftrightarrow[\frac{\tau_3}{2}+\tau_4]{\lambda_{v_3,v_4}} (v_4)$ as a compromised node, as shown in line 22 of Algorithms 3 and line 19 of Algorithm 4, respectively. That is, the packets' delays at this hop, $\frac{\tau_3}{2} + \tau_4$, do not match its announced contact frequency, λ_{v_3,v_4}. Accordingly, if Eve compromises n adjacent nodes, $v_1, v_2, .., v_n, .., d$ in a path, she is able to launch fake PRTs attack by faking the PRTs among $v_1, v_2, .., v_{n-1}$ nodes, however, the last compromised node by Eve, v_n, will be detected either by the ultimate destination node, d, in case of HDT or by the first non-compromised node, v_{n+1}, in case of EHDT.

3) False positive rate for dynamic ICTs. We analyzed the impact of dynamic scenarios in OMNs on the false positive rates of PDT, HDT, and EHDT. That is, when the ICTs are dynamic and frequently changing over time (e.g., due to high-mobility of nodes). We repeated the same experiments steps that were presented at the beginning of Section 6.2 for both Reality and UCSD traces. However, instead of randomly extracting 1,000 paths during the first step, we performed the following steps to select 1,000 dynamic paths (the paths that have dynamic ICTs amongst their nodes). First, we calculated the Standard Deviation (STDEV) values of the ICTs amongst all pairs of nodes for both traces, as shown in Figures 15(a) and 16(a). The figures represent the bar graphs of all STDEV values of ICTs (sorted ascendingly). We disregarded all pair of nodes if the STDEV of its nodes' ICTs is 0.0. Second, we determined S_{dyn}, a set of pairs of nodes that contains the top 75 percentile of the STDEV's

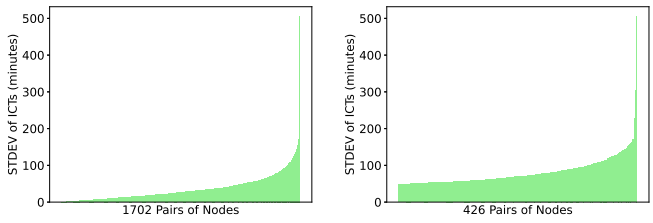


Fig. 15: STDEV of ICTs for Reality trace for (a) All pairs of nodes [0.008, 506.3] minutes. (b) Upper quartile (STDEV of ICT $\geq 75^{th}$ percentile) [47.6, 506.3] minutes.

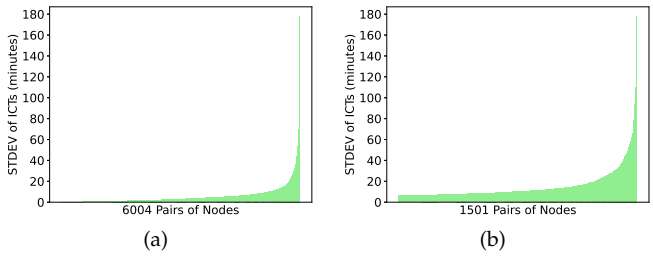


Fig. 16: STDEV of ICTs for UCSD trace for (a) All pairs of nodes [0.006, 178.1] minutes. (b) Upper quartile (STDEV of ICT $\geq 75^{th}$ percentile) [6.2, 178.1] minutes.

for both traces, as shown in Figures 15(b) and 16(b). Third, we selected 1,000 dynamic paths such that all their nodes $\in S_{dyn}$ and calculated the false positive rates of PDT, HDT, and EHDT. As shown in Figures 17(a) and 17(b), the false positive rates increase when the ICTs are dynamic and change frequently in OMNs. However, as HDT and EHDT perform hop-wise detection, they incur less impact on their false positive rates (compared to PDT).

4) Robustness against short ICTs. We assumed in Section 2.1 that it is sufficient for PDT, HDT, and EHDT to maintain a scale of one second time synchronization between OMNs nodes (the ICTs in OMNs are usually at the scale of seconds/minutes). In this section, we aim to evaluate the robustness of our approaches against shorter ICTs and analyze their impact on the detection rates. We repeated the same experiments steps that were presented at the beginning of Section 6.2 for both Reality and UCSD traces. However, instead of randomly extracting 1,000 paths during the first step, we performed the following steps to select 1,000 paths with the shortest ICTs. Specifically, the paths where their nodes have to be timely-synchronized with the tightest/minimum thresholds. First, we calculated the average values of the ICTs amongst all pairs of nodes for both traces, as shown in Figures 18(a) and 19(a) (the bar graphs of average ICTs values sorted ascendingly). Second, we determined $S_{shortICTs}$, a set of pairs of nodes that contains the bottom 5 percentile of ICTs for both traces (Figures 18(b) and 19(b)). The shortest ICTs for both traces are 92.0 and 0.96 seconds, which validate our assumption about the scale of ICTs. Third, we selected 1,000 paths such that all their nodes $\in S_{shortICTs}$ and calculated the detection rates of PDT (when the ratio of the compromised links = $16.6\%(\frac{1}{6})$), HDT, and EHDT. As shown in Figures 20(a), 20(b), 21(a),

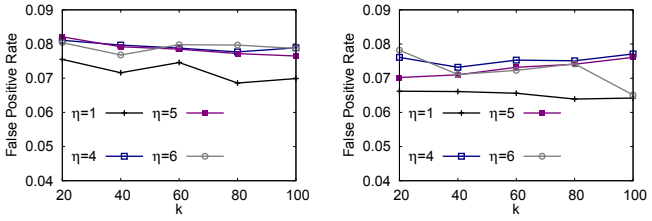


Fig. 17: False Positive Rate for dynamic ICTs scenarios for (a) Reality trace of PDT (η = 4, 5, 6 hops) and of HDT and EHDT (η = 1 hop). (b) UCSD trace of PDT (η = 4, 5, 6 hops) and of HDT and EHDT (η = 1 hop).

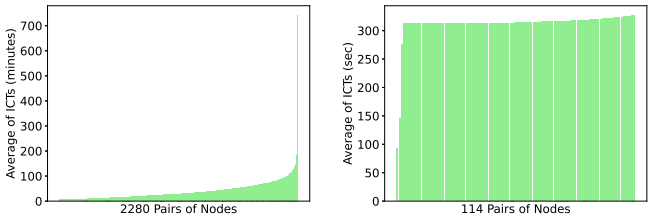


Fig. 18: Average ICTs for Reality trace for (a) All pairs of nodes [1.5, 742.6] minutes. (b) Pairs of nodes with ICT $\leq 5^{th}$ percentile [92.0, 327.5] sec.

and 21(b), the detection rates have not crucially degraded and still comparable to the detection rates shown in Figures 6(a), 6(h), 10(a), and 10(h), respectively.

We also evaluated the impact of short ICTs on the detection rates of PDT, HDT, and EHDT on our Asus notebooks testbed by repeating the same experiments steps that we presented in Section 3.1. However, for the second step, we set the on-off ratios to 1.0 (theoretically represent fully-connected OMN with ICT= 0.0 seconds) and calculated the detection rates of PDT for HRP and Prophet protocols. As shown in Figures 22(a) and 22(b), the detection rates slightly decrease compared to Figures 12(a), 12(b), 12(c), and 12(d) for 0.4 and 0.9 on-off ratios, respectively. However, the detection rates still increase while increasing the number of compromised links and received packets ($\geq 90\%$ for $k \geq 80$).

5) Feasibility of recovering the compromised nodes' private keys. We assumed in Section 2.2 that Eve is able to compromise the victim nodes' private keys. We aim here to briefly discuss the feasibility of our assumption. Some recent works discussed the task of recovering the private keys of RSA and Elliptic Curve Digital Signature Algorithms (ECDSA) [72], [73]. It's shown in [74], [75] that the private keys of RSA can be recovered from noisy key bits with erasures and errors and by using special partial information. Other approaches [76]–[78] recover the RSA private keys from a TLS session, by factorizing the widely used RSA moduli (i.e., passive RSA key recovery), or by abusing translation look-aside buffers (TLBs) information. For ECDSA, Hidden Number Problem (HNP) [73] was used in recovering the ECDSA private keys in many attacks against biased/leaked nonces by utilizing side channels such as timing [79], [80] or cache attacks [81]–[83]. Moreover, by using information about nonce distribution [84] or fault injection [85].

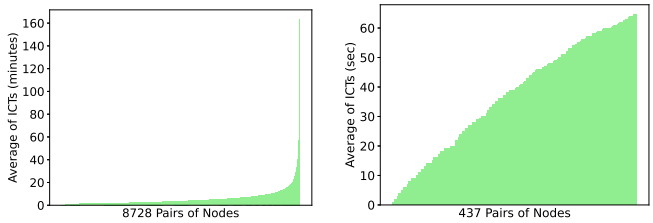


Fig. 19: Average ICTs for UCSD trace for (a) All pairs of nodes [0.016, 163.4] minutes. (b) Pairs of nodes with ICT $\leq 5^{th}$ percentile [0.96, 64.6] sec.

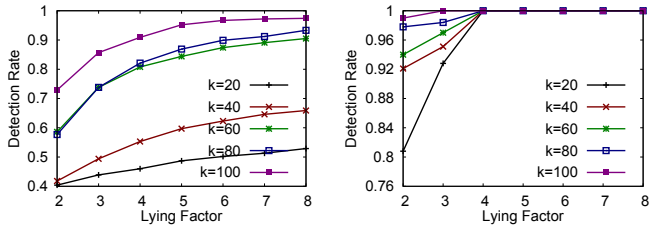


Fig. 20: Detection Rate for paths with ICTs $\leq 5^{th}$ percentile for Reality trace for: (a) PDT's when the ratio of the compromised links $16.6\%(\frac{1}{6})$. (b) HDT and EHDT.

8 State of the Art

Previous works on the security threats in OMNs focused on jamming, blackhole, flood, wormhole, and packet dropping attacks as well as on authenticating the nodes, trust-management, and privacy-preserving [22]–[35]. In [22], a statistical-based jamming attack detection approach was proposed by leveraging the collected statistical measures from the relay nodes and a prescribed packet delivery ratio threshold. The adversary in the flood attack floods junk data into the network in order to deplete or overuse the limited network resources. The nodes in [25] employs rate limiting to defend against flood attacks. That is, each node has limits for the total number of packets and replicas that it can generate in each time interval. The adversary in the wormhole attack records the packets at one location and tunnels them to another colluding node to corrupt the topology views of the network. In order to address the wormhole attack, a detection mechanism is proposed [26] to reduce nodes' transmission range for a short time to detect the presence of a forbidden topology structure that is caused by a wormhole attack. In the packet dropping attack, the adversary intentionally drops all or part of the received packets. A machine learning method based on utilizing classifiers to improve routing and detect the blackhole attack was proposed in [23]. The nodes in [24] [27] exchange signed contact records, based in which the next contact nodes can detect if the attacker has dropped any packet. In [28], a packet dropping is detected and traced back based on the Merkle tree. Indeed, none of the aforementioned works address the route hijacking attack in OMNs. However, most

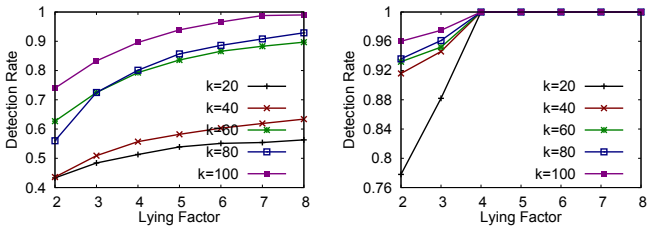


Fig. 21: Detection Rate for paths with ICTs $\leq 5^{th}$ percentile for UCSD trace for: (a) PDT's when the ratio of the compromised links $16.6\%(\frac{1}{6})$. (b) HDT and EHDT.

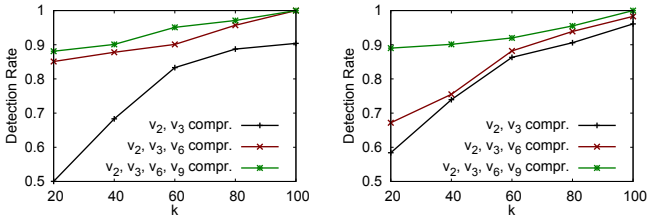


Fig. 22: PDT's Detection Rate on Asus notebooks testbed using 1.0 on-off ratio for: (a) HRP (b) Prophet.

of route hijacking attacks that have been addressed by recent research are in Internet. Each Autonomous System (AS) in Internet manages a number of networks, which can be expressed as IP prefixes. ASes use BGP to advertise their IP prefixes and establish inter-domain routes in Internet. BGP is a distributed protocol, lacking authentication of routes. Hence, a malicious AS can claim to own a prefix or sub-prefix that belongs to another AS causing redirection of routes from that AS to the attacker.

BGP hijacking detection approaches can be classified into four categories. *Control-plane approaches* [36]–[40] collect BGP updates or routing tables from a distributed set of public BGP monitoring infrastructure and route collectors such as [41]–[43], and raise alarms when a change in the origin-AS of a prefix, or a suspicious route is observed. PHAS and Cyclops [38] [39] are notification systems that alert prefix owners (i.e., ISPs) when their BGP origin change. The network administrator in ARTEMIS [40] stores a configuration file that has an up-to-date list of all owned and announced prefixes. This list is continuously compared with the collected BGP updates from the monitoring services (i.e. [41]–[43]). Based on the result of the comparison, ARTEMIS can detect any hijacking event and generate alerts accordingly. The aforementioned control-plane approaches cannot be used against the CollusiveHijack attack. Apart from the fact that there are no network administrators in OMNs, the nodes cannot create a list of "owned" or reached IP's *a priori* (i.e., the nodes do not know the future contacts among each other). *Data-plane approaches* [44] [45] continuously probe Internet to detect whether any data path changes. That is, by using pings/traceroutes to monitor the connectivity of a prefix and raise an alarm, when significant changes in the reachability [44] of a prefix or the paths leading to it [45] are observed. The Listen protocol [86] is a data-plane verification technique that detects reachability problems in the data plane by passively probing network and checking whether the underlying routes to different destinations work. Due to the intermittent connectivity between the nodes in OMNs, data-plane approaches fail and cannot be used to detect the CollusiveHijack attack. *Hybrid approaches* [87]–[89] combine control and data plane information to detect the hijacking attack, however, they cannot detect the CollusiveHijack attack because they still need network administrators [89] and use monitor tools like pings and traceroutes [88] [87]. *Cryptographic approaches* [90]–[92] use PKI to ensure the authentication of routing announcements to minimize the risk of a single non-colluding hijacking (cannot defend against colluding ASes). S-BGP [92] and Whisper protocol [86] fail to detect colluding ASes that have a direct link to tunnel packets/advertisements unless the complete topology of the network is known and enforced. However, the topology of OMNs is dynamic. The detection of colluding ASes is beyond the scope of the BGPsec protocol [90].

Other works focused on authenticating the nodes in OMNs [32]–[35]. An aggregate signature-based authentication scheme is proposed in [32] to effectively and efficiently process the exchanged messages amongst the nodes in OMNs to address the rogue pubic key, forgery, eavesdropping, replay, and man-in-the-middle attacks. In [33], the security features of the NTRU algorithm (i.e., an asymmetric post-quantum cryptosystem algorithm) are leveraged to propose an authentication scheme that generates unique IDs, encryption keys, and decryption keys for the nodes in OMNs. In [34], a randomized light-weight authentication protocol that involves node registration and authentication phases using identity-based encryption (IBE) scheme is proposed. The proposed protocol [34] aims to generate public keys from publicly available nodes with high trust values (instead of only generating the keys from a central registration server). [35] proposes a new protocol that is called message trust-based secure multipath routing protocol (MT-SMRP). MT-SMRP aims to route the packets through disjoint paths in which each of them applies a soft-encryption technique to prevent packet fabrication attacks in OMNs.

9 Conclusions and Future Works

We presented three algorithms, PDT, HDT, and EHDT, to detect the CollusiveHijack attack in OMNs. PDT, HDT, and EHDT employ the KS2ST and offer a trade-off between the compatibility with BSP, the detection rate, and the detection latency against the CollusiveHijack attack. Moreover, EHDT crucially enhances the detection capabilities against the CollusiveHijack attack by enabling the intermediate nodes in OMNs to detect the compromised nodes before the packets' destination nodes (as is the case of PDT and HDT). Our evaluation results show that the proposed algorithms are able to identify the CollusiveHijack attack with ∼80% and 99.4% detection rates, while maintaining ∼3.6% false positive rate, and with short detection latency (7-14 hours) for EHDT.

In our future work, we plan to investigate the capability of the attacker, Eve, to fake the ICTs in order to launch a

Denial-of-Service (DoS) in OMNs. That is, when Eve lies about her nodes' ICTs with the goal of avoiding receiving/routing the data packets of the legitimate nodes in OMNs by claiming that her nodes meet less frequently than in reality. Eve aims to deceive HRP and Prophet protocols and to prevent the data packets to be delivered to the legitimate nodes in OMNs. To the best of our knowledge, the aforementioned DoS attack has neither been identified nor addressed in OMNs as most of the previous research addressed flood or packet dropping DoS attacks as well as the selfish nodes in OMNs [93]–[95].

APPENDIX

MATHEMATICAL ANALYSIS

We provide mathematical argument and analysis to demonstrate the guarantees of the detection capabilities of PDT, HDT, and EHDT that leverage the Kolmogorov-Smirnov two-sample test (KS2ST). We prove that when the random samples (i.e., the packets' delays in our case) are in fact coming from the corresponding hypo-exponential distribution, there is no difference detected by the KS2ST. Subsequently, it is shown that when the random samples are from different hypo-exponential distributions and in the presence of large number of samples, then, the difference will be detected.

We denote, F_0 : The hypo-exponential distribution with parameters $\lambda_{i,1}, \lambda_{1,2}, \ldots, \lambda_{\eta-1,j}$. Suppose that $x_1, x_2, \ldots, x_n$ are the random samples with $n = 20, 40, 60, 80,$ and 100. The empirical distribution of $x_1, x_2, \ldots, x_n$ is denoted as $\hat{F}_n(.)$ and is defined by:

$$\hat{F}_n(x) = \frac{1}{n}\sum_{i=1}^{n} I_{[x_i,\infty)}(x),$$

where $I_{\mathcal{S}}(.)$ is the indicator function for the set $\mathcal{S}$. The following result establishes that when the size of the random sample is sufficiently large, the sampling distribution of the random sample converges to the data generating hypo-exponential distribution.

Theorem 1. $||\hat{F}_n - F_0||_\infty = sup_{x\in\mathbb{R}}|\hat{F}_n(x) - F_0(x)| \xrightarrow{a.s.} 0.$

In the following, we sketch the proof using the results from [96].

Proof. Given $\epsilon > 0$, there exists a partition of $\mathbb{R}$, $-\infty = x_0 < x_1 < \ldots < x_k = \infty$ such that for $x \in (x_{i-1}, x_i)$

$$\hat{F}_n(x) - F_0(x) \leq \lim_{x\uparrow x_i} \hat{F}_n(x) - \lim_{x\uparrow x_i} F_0(x) + \epsilon,$$

and

$$\hat{F}_n(x) - F_0(x) \geq \lim_{x\downarrow x_{i-1}} \hat{F}_n(x) - \lim_{x\downarrow x_{i-1}} F_0(x) - \epsilon.$$

Now, since $\hat{F}_n(x) \xrightarrow{a.s.} F(x)$, this implies that for a given $\epsilon > 0$ and $x \in (x_{i-1}, x_i)$, there exists an integer N such that for all $n \geq N$, $|\hat{F}_n(x) - F(x)| < 2\epsilon$ i.e., $\hat{F}_n(x)$ converges uniformly to $F(x)$ with probability one. This uniform convergence implies that

$$\Pr(\lim_{n\to\infty} \sup_{x\in\mathbb{R}} |\hat{F}_n(x) - F_0(x)| = 0) = 1,$$

or equivalently

$$||\hat{F}_n - F_0||_\infty \xrightarrow{a.s.} 0.$$

□

The above result suggests that the test statistics

$$D_n = \sup_x |\hat{F}_n(x) - F_0(x)|$$

will be useful in detecting the differences in distribution between the random sample and the corresponding hypo-exponential distribution. The distribution of D_n does not depend on F and hence is non-parametric. Accordingly, the exact analytical form can not be obtained; as such, various approximations are given in the literature, for example, see [97], [98]. Using the approximations, the 5% critical value is obtained to be 1.36. That is, if the estimated value of D_n is more than $1.36/\sqrt{n}$ or more than 0.0136, then, we reject the null hypothesis that the random sample is generated from the corresponding hypo-exponential distribution.

In the next result, we establish that when the edges $\lambda_{i,1}, \lambda_{1,2}, \ldots, \lambda_{\eta-1,j}$ are multiplied by some factor (i.e., Definition 2.1, when Eve claims that her compromised nodes meet more frequently than in reality) and as a result the realization is in fact not from a hypo-exponential distribution $F_0(.)$. Then, the probability of rejecting the null hypothesis that the sample size is not from $F_0(.)$ approaches to one. We begin by denoting the alternative distribution by $F_1(.)$

In the sequel, we compute the probability when $x_1, x_2, \ldots, x_n$ are sampled from $F_1(.)$, then, the null hypothesis $H_0 : F_1(.) = F_0(.)$ is rejected using the notions from [99]. Recognize that the probability under consideration is in fact power of the underlying statistical test. Also note that the probability of rejecting the null hypothesis is quantified by $\Pr(D_n > d_{\alpha;n}/\sqrt{n})$, where $d_{\alpha;n}$ is the critical value of approximate Kolmogorov-Smirnov distribution at level α. For our case, α = 0.05, and $d_{\alpha;n} = 1.36$.

Theorem 2.

$$\lim_{n\to\infty} \Pr(D_n > d_{\alpha;n}/\sqrt{n}) = 1$$

Proof. Let $\Delta = \max_x |F_1(x) - F_0(x)|$ and let x_0 be a value of x such that $\Delta = |F_1(x_0) - F_0(x_0)|$. Then, $\Pr(D_n > d_{\alpha;n}) > 1 - \Pr(F_0(x_0) - \frac{d_{\alpha;n}}{\sqrt{n}} < D_n(x_0) < F_0(x_0) + \frac{d_{\alpha;n}}{\sqrt{n}}$.

Replacing $F_0(x_0) = F_1(x_0) \pm \Delta$ we obtain

$$\Pr(D_n > d_{\alpha;n}) > 1 - \Pr(\frac{-d_{\alpha;n} \pm \Delta\sqrt{n}}{\sqrt{F_1(x_0)(1-F_1(x_0))}} < \frac{D_n(x_0) - F_1(x_0)\sqrt{n}}{\sqrt{F_1(x_0)(1-F_1(x_0))}} < \frac{d_{\alpha;n} \pm \Delta\sqrt{n}}{\sqrt{F_1(x_0)(1-F_1(x_0))}}).$$

For sufficiently large n, $-d_{\alpha;n} \pm \Delta\sqrt{n}$ and $d_{\alpha;n} \pm \Delta\sqrt{n}$ will have same sign and $F_1(x_0)$ can be replaced with $\frac{1}{2}$. With these ingredients, and for large n, using a Normal approximation, we get

$$\Pr(D_n > d_{\alpha;n}) > 1 - \int_{2(-d_{\alpha;n}\pm\Delta\sqrt{n})}^{2(d_{\alpha;n}\pm\Delta\sqrt{n}} \phi(t)dt$$

TABLE 1: Minimum number of data packets, k, to detect the maximum difference between the hypo-exponential distribution and the sample distribution.

Max. Absolute Difference	Min. k
0.14	94.4
0.15	82.2
0.16	72.2
0.17	64.0
0.18	57.1
0.19	51.2
0.20	46.2
0.25	29.6
0.30	20.5

where $\phi(t)$ is the density function of the standard Normal distribution. Note that, as $n \to \infty$, the integral tends to zero which completes the proof. □

In order to statistically investigate the required number of received packets, k, that are needed for the KS2ST to detect the CollusiveHijack attack via PDT, HDT, and EHDT. We refer the readers to Figure 2 in [100], which depicts the lower bound for the power of the Kolmogorov-Smirnov test. Note that the actual power is higher in practice. For our case, we consider the curve for $\alpha = 0.05$. Hence, the critical value at this point is 1.36 (discussed at the beginning of this section). Now, when the maximum absolute difference between the hypo-exponential distribution and the sampling distribution of the random samples is 0.14, to obtain a test with power more than 0.5 we need to solve the equation $0.14\sqrt(n) = 1.36$ (n is the required number of packets to detect the 0.14 difference). This gives rise the required number of packets as at least 94. Similarly, when the maximum absolute differences are higher, less number of packets are required to detect such differences. In Table 1, we provide the minimum number of k to detect for some values of the maximum absolute difference. We note that when the maximum absolute difference is 0.3, then, 20 packets postulates at least 0.5 power. In our work, we conducted experiments with different number of packets (20, 40, 60, 80, 100) to capture a wide range of existing differences. As shown in Figures 6(a)-6(g) and 10(a)-10(g), PDT achieved $\geq$ 90.0% detection rates for most of the cases (when Eve hijacks $\geq$ 60 packets). However, for HDT and EHDT, the detection rates (when Eve hijacks $\geq$ 60 packets) were $\geq$ 99.0% (Figures 6(h) and 10(h)). We acknowledge that the detection rates of our approaches decreases when $k < 60$ and Eve might not be detected in that case. Specifically, if Eve changes her behaviour (e.g., stops lying about the ICTs of her compromised nodes) before k becomes ≥ 60.

ACKNOWLEDGMENT

This work was supported in part by the National Institute of Standards and Technology (NIST) under grant NO. (#70NANB17H190). We would like to thank the anonymous reviewers for their insightful comments and suggestions.

REFERENCES

[1] Distressnet-ng project. http://distressnet.net/.

[2] S. Datta and S. Madria. Efficient photo crowdsourcing with evolving pois under delay-tolerant network environment. *Pervasive and Mobile Computing*, 67, 2020.

[3] Firechat: Google play store. https://play.google.com/store.

[4] 1am: Censorship-resistant microblogging. http://1am-networks.org.

[5] E. Harkavy and M. S. Net. Utilizing reinforcement learning to autonomously mange buffers in a delay tolerant network node. *AeroConf'20*.

[6] E. Tikhonov, D. Schneps-Schneppe, and D. Namiot. Delay tolerant network protocols for an expanding network on a railway. *3ICT'20*.

[7] J. Wu, Y. Guo, H. Zhou, L. Shen, and L. Liu. Vehicular delay tolerant network routing algorithm based on bayesian network. *IEEE Access*, 8:18727–18740, 2020.

[8] Y. Dong, F. Zhang, I. Joe, H. Lin, W. Jiao, and Y. Zhang. Learning for multiple-relay selection in a vehicular delay tolerant network. *IEEE Access*, 8:175602–175611, 2020.

[9] E. Yaacoub, K. Abualsaud, T. Khattab, and A. Chehab. Secure transmission of iot mhealth patient monitoring data from remote areas using dtn. *IEEE Network*, 34(5):226–231, 2020.

[10] C. Yang and R. Stoleru. Hybrid routing in wireless networks with diverse connectivity. *MobiHoc'16*.

[11] A. Lindgren, A. Doria, E. Davies, and O. Schelén. Probabilistic routing in intermittently connected networks. *SIGMOBILE Mob. Comput. Commun. Rev.*, 7(3), 2003.

[12] A. Lindgren, A. Doria, E. Davies, and S. Grasic. Probabilistic routing protocol for intermittently connected networks. irtf rfc 6693. 2012.

[13] X. Tie, A. Venkataramani, and A. Balasubramanian. R3: Robust replication routing in wireless networks with diverse connectivity characteristics. *MobiCom'11*.

[14] A. Balasubramanian, B. Levine, and A. Venkataramani. Dtn routing as a resource allocation problem. *SIGCOMM'07*.

[15] N. Eagle and A. Pentland. Reality mining: Sensing complex social systems. *Personal Ubiquitous Comput.*, 10(4), 2006.

[16] Marvin McNett and Geoffrey M. Voelker. Access and mobility of wireless pda users. *ACM SIGMOBILE Mobile Computing and Communications Review*, 9(2), 2005.

[17] R. Patil and M. P. Tahiliani. Detecting packet modification attack by misbehaving router. *ICNSC'14*.

[18] J. Burgess, G. D. Bissias, M. Corner, and B. N. Levine. Surviving attacks on disruption-tolerant networks without authentication. *MobiHoc'07*.

[19] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic. Robust smartphone app identification via encrypted network traffic analysis. *IEEE TIFS*, 13(1), 2018.

[20] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic. Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic. *EuroS&P'16*.

[21] R. Lu, X. Lin, H. Zhu, X. Shen, and B. Preiss. Pi: A practical incentive protocol for delay tolerant networks. *IEEE TWC*, 9(4), 2010.

[22] J. Singh, I. Woungang, S. K. Dhurandher, and K. Khalid. A jamming attack detection technique for opportunistic networks. *Internet of Things*, 17, 2022.

[23] Y. Gao, J. Tao, Y. Xu, Z. Wang, W. Sun, and G. Cheng. Cebd: Contact-evidence-driven blackhole detection based on machine learning in oppnets. *IEEE Transactions on Computational Social Systems*, 8(6):1344–1356, 2021.

[24] F. Li, J. Wu, and A. Srinivasan. Thwarting blackhole attacks in disruption-tolerant networks using encounter tickets. *INFOCOM'09*.

[25] Q. Li, W. Gao, S. Zhu, and G. Cao. To lie or to comply: Defending against flood attacks in disruption tolerant networks. *IEEE TDSC*, 10(3), 2013.

[26] Y. Ren, M. C. Chuah, J. Yang, and Y. Chen. Detecting wormhole attacks in delay-tolerant networks. *IEEE Wireless Comm.*, 17(5), 2010.

[27] Q. Li and G. Cao. Mitigating routing misbehavior in disruption tolerant networks. *IEEE TIFS*, 7(2), 2012.

[28] M. Alajeely, R. Doss, A. Ahmad, and V. Mak-Hau. Defense against packet collusion attacks in opportunistic networks. *Computers and Security*, 65, 2017.

[29] M. Abouarork and K. Ahmad. Authentication in opportunistic networks: State and art. *Journal of Discrete Mathematical Sciences and Cryptography*, pages 1–12, 2021.

[30] E.K. Wang, Y. Li, Y. Ye, S. M. Yiu, and L. C. K. Hui. A dynamic trust framework for opportunistic mobile social networks. *IEEE Transactions on Network and Service Management*, 15(1):319–329, 2018.

[31] S. Rashidibajgan, T. Hupperich, R. Doss, and A. Forster. Secure and privacy-preserving structure in opportunistic networks. *Computers & Security*, 104, 2021.

[32] C. B. Avoussoukpo, C. Xu, M. Tchenagnon, and N Eltayieb. Towards an aggregate signature-based authentication for opportunistic networks. *CyberSA'20*.

[33] M. Abouaroek and K. Ahmad. Node authentication using ntru algorithm in opportunistic network. *Scalable Computing: Practice and Experience*, 20, 2019.

[34] K. Wang and K. Sakai. Randomized authentication using ibe for opportunistic networks. *ICPP Workshops'20*.

[35] S. Dhurandher, J. Singh, I. Woungang, R. Kumar, and G Gupta. Message trust-based secure multipath routing protocol for opportunistic networks. *International Journal of Communication Systems*, 33, 2020.

[36] Bgpmon. http://www.bgpmon.net.

[37] Ripe myasn system. http://www.ris.ripe.net/myasn.html.

[38] M. Lad, D. Massey, D. Pei, Y. Wu, B. Zhang, and L. Zhang. Phas: A prefix hijack alert system. *USENIX'06*.

[39] Y. Chi, R. Oliveira, and L. Zhang. Cyclops: The as-level connectivity observatory. *ACM SIGCOMM Computer Comm. Review*, 38(5), 2008.

[40] P. Sermpezis, V. Kotronis, P. Gigis, X. Dimitropoulos, D. Cicalese, A. King, and A. Dainotti. Artemis: neutralizing bgp hijacking within a minute. *Technical Report'18*.

[41] Bgpmon (colorado state university). https://www.bgpmon.io/.

[42] Routing information service (ris). https://www.ripe.net/.

[43] Route views project(university of oregon). http://www.routeviews.org/.

[44] Z. Zhang, Y. Zhang, Y. Charlie Hu, Z. Morley Mao, and R. Bush. ispy: Detecting ip prefix hijacking on my own. *IEEE/ACM TN*, 18(6), 2010.

[45] C. Zheng, L. Ji, D. Pei, J. Wang, and P. Francis. A light-weight distributed scheme for detecting ip prefix hijacks in real-time. *SIGCOMM'07*.

[46] J. Frank and Jr. Massey. The kolmogorov-smirnov test for goodness of fit. *Journal of the American Statistical Association*, 46(253), 1951.

[47] S. Gianvecchio and H. Wang. An entropy-based approach to detecting covert timing channels. *IEEE TDSC*, 8(6), 2011.

[48] A. L. Toledo and X. Wang. Robust detection of selfish misbehavior in wireless networks. *IEEE JSAC*, 25(6), 2007.

[49] J. Tapiador, P. Teodoro, and J. Verdejo. Measuring normality in http traffic for anomaly-based intrusion detection. *Computer Networks*, 45(2), 2004.

[50] J. Caberera, B. Ravichandran, and R. Mehra. Statistical traffic modeling for network intrusion detection. *MASCOTS'00*.

[51] S. Symington, S. Farrell, H. Weiss, and P. Lovell. Bundle security protocol specification. https://tools.ietf.org/html/rfc6257. 2011.

[52] A. Altaweel, R. Stoleru, G. Gu, and A. Maity. Collusivehijack: A new route hijacking attack and countermeasures in opportunistic networks. *CNS'19*.

[53] C. Yang and R. Stoleru. Hybrid routing in wireless networks with diverse connectivity. *Technical Report, Texas A&M University*.

[54] Ibr-dtn - a modular and lightweight implementation of the bundle protocol. https://github.com/ibrdtn/ibrdtn.

[55] M. Doering, S. Lahde, J. Morgenroth, and L. Wolf. Ibr-dtn: An efficient implementation for embedded systems. *CHANTS'08*.

[56] N. Srinidhi, C. Sagar, J. Shreyas, and D. Kumar SM. An improved prophet-random forest based optimized multi-copy routing for opportunistic iot networks. *Internet of Things*, 11, 2020.

[57] M. W. Kang, D. Y. Seo, and Y. W. Chung. An efficient opportunistic routing protocol for icn. *ICN'19*.

[58] Y. Mao, C. Zhou, Y. Ling, and J. Lloret. An optimized probabilistic delay tolerant network (dtn) routing protocol based on scheduling mechanism for internet of things (iot). *Sensors*, 19(2):243, 2019.

[59] K. M. Baek, D. Y. Seo, and Y. W. Chung. An improved opportunistic routing protocol based on context information of mobile nodes. *Applied Sciences*, 8(8), 2018.

[60] D. K. Sharma, S. K. Dhurandher, I. Woungang, R. K. Srivastava, A. Mohananey, and J. J. P. C. Rodrigues. A machine learning-based protocol for efficient routing in opportunistic networks. *IEEE Systems Journal*, 12(3):2207–2213, 2018.

[61] S. Basu, A. Biswas, S. Roy, and S. DasBit. Wise-prophet: a watchdog supervised prophet for reliable dissemination of post disaster situational information over smartphone based dtn. *Journal of Network and Computer Applications*, 109:11–23, 2018.

[62] S. J. Borah, S. K. Dhurandher, S. Tibarewala, I. Woungang, and M. S. Obaidat. Energy-efficient prophet-prowait-edr protocols for opportunistic networks. *GLOBECOM'17*.

[63] A. Keränen, J. Ott, and T. Kärkkäinen. The one simulator for dtn protocol evaluation. *Simutools'09*.

[64] K. Smaili, T. Kadri, and S. Kadry. Hypoexponential distribution with different parameters. *Applied Mathematics*, 4(4), 2013.

[65] J. C. Ferreira and C. M. Patino. What does the p value really mean? *Jornal Brasileiro de Pneumologia*, 41(5), 2015.

[66] The r project for statistical computing. https://www.r-project.org/.

[67] Rcpp for seamless r and c++ integration. http://www.rcpp.org/.

[68] B. Lynn. The pairing-based cryptography (pbc) library. https://crypto.stanford.edu/pbc/.

[69] F. Hess. Efficient identity based signature schemes based on pairings. *SAC'02*.

[70] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot. Optimized link state routing protocol for ad hoc networks. *INMIC'01*.

[71] A. Tonnesen. Implementing and extending the optimized link state routing protocol. *Masters thesis, University of Oslo'04*.

[72] G. D. Micheli and N. Heninger. Recovering cryptographic keys from partial information, by example. 2020. working paper or preprint.

[73] J. Jancar, V. Sedlacek, P. Svenda, and M. Sys. Minerva: The curse of ecdsa nonces : Systematic analysis of lattice attacks on noisy leakage of bit-length of ecdsa nonces. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(4), 2020.

[74] C. Krebs. Algorithms for rsa key recovery. *Masters thesis, Institute for IT Security, The University of Lbeck*, 2021.

[75] C. Patsakis. Recovering rsa private keys on implementations with tampered lsbs. *SECRYPT'13*.

[76] M. Ortisi. Recover a rsa private key from a tls session with perfect forward secrecy. https://www.blackhat.com/. 2016.

[77] M. Nemec, M. Sýs, P. Svenda, D. Klinec, and V. Matyas. The return of coppersmith's attack: Practical factorization of widely used rsa moduli. *CCS'17*.

[78] B. Gras, K. Razavi, H. Bos, and C. Giuffrida. Translation leak-aside buffer: Defeating cache side-channel protections with TLB attacks. *USENIX Security'18*.

[79] A. C. Aldaya, B. B. Brumley, S. ul Hassan, C. P. Garca, and N. Tuveri. Port contention for fun and profit. *IEEE SP'19*.

[80] D. Moghimi, B. Sunar, T. Eisenbarth, and N. Heninger. Tpm-fail: Tpm meets timing and lattice attacks. *USENIX'20*, 2020.

[81] K. Ryan. Hardware-backed heist: Extracting ecdsa keys from qualcomm's trustzone. *CCS'19*.

[82] K. Ryan. Return of the hidden number problem.: A widespread and novel key extraction attack on ecdsa and dsa. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(1):146–168, Nov. 2018.

[83] S. Weiser, D. Schrammel, L. Bodner, and R. Spreitzer. Big numbers - big troubles: Systematically analyzing nonce leakage in (EC)DSA implementations. *USENIX Security'20*.

[84] J. Breitner and N. Heninger. Biased nonce sense: Lattice attacks against weak ecdsa signatures in cryptocurrencies. *IACR Cryptol. ePrint Arch.*, 2019:23, 2019.

[85] M. Liu, J. Chen, and H. Li. Partially known nonces and fault injection attacks on sm2 signature algorithm. *ICISC'13*.

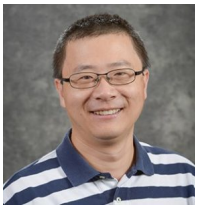
[86] L. Subramanian, V. Roth, I. Stoica, S. Shenker, and R. H. Katz. Listen and whisper: Security mechanisms for bgp. *NSDI'04*.

[87] X. Hu and Z. M. Mao. Accurate real-time identification of ip prefix hijacking. *S&P'07*.

[88] X. Shi, Y. Xiang, Z. Wang, X. Yin, and J. Wu. Detecting prefix hijackings in the internet with argus. *IMC'12*.
[89] J. Schlamp, R. Holz, Q. Jacquemart, G. Carle, and E. W. Biersack. Heap: Reliable assessment of bgp hijacking attacks. *IEEE JSAC*, 34(6), 2016.
[90] M. Lepinski and K. Sriram. Bgpsec protocol specification. https://tools.ietf.org/html/rfc8205. 2017.
[91] M. Lepinski and S. Kent. An infrastructure to support secure internet routing. https://tools.ietf.org/html/rfc6480. 2012.
[92] S. Kent, C. Lynn, and K. Seo. Secure border gateway protocol (s-bgp). *IEEE JSAC*, 18(4), 2000.
[93] W. Khalid, N. Ahmed, M. Khalid, A. Ud Din, A. Khan, and M. Arshad. Frid: Flood attack mitigation using resources efficient intrusion detection techniques in delay tolerant networks. *IEEE Access*, 7, 2019.
[94] T. N. D. Pham, C. K. Yeo, N. Yanai, and T. Fujiwara. Detecting flooding attack and accommodating burst traffic in delay-tolerant networks. *IEEE Transactions on Vehicular Technology*, 67(1), 2018.
[95] W. Khalid, Z. Ullah, N. Ahmed, Y. Cao, M. Khalid, M. Arshad, F. Ahmad, and H. Cruickshank. A taxonomy on misbehaving nodes in delay tolerant networks. *Computers & Security*, 77, 2018.
[96] A. M. Polansky. *Introduction to statistical limit theory*. International series of monographs on physics. Chapman and Hall/CRC, 2011.
[97] J.D. Gibbons and S. Chakraborti. *Nonparametric Statistical Inference: Revised and Expanded*. CRC Press, 2014.
[98] H. Niederhausen. Sheffer polynomials for computing exact kolmogorov-smirnov and renyi type distributions. *The Annals of Statistics*, 9(5):923–944, 1981.
[99] F. J. Massey Jr. A note on the power of a non-parametric test. *The Annals of Mathematical Statistics*, 21(3):440–443, 1950.
[100] F. J. Massey. The kolmogorov-smirnov test for goodness of fit. *Journal of the American Statistical Association*, 46(253):68–78, 1951.

Dr. Ala Altaweel is currently an Assistant Professor in the Department of Computer Engineering at the University of Sharjah. Dr. Ala's research interests are in cyber security, wireless networks and computer security, distributed systems, and edge computing. Dr. Ala received his Ph.D. in Computer Engineering from Texas A&M University, USA, in 2019. In 2009 and 2006, Dr. Ala received his M.S. in Information Technology from the University of Stuttgart, Germany, and his B.Sc. in Computer Engineering from Jordan University of Science and Technology, Jordan, respectively. Before joining the University of Sharjah, he worked as a Postdoctoral Researcher in the Laboratory for Embedded & Networked Sensor Systems at Texas A&M University. Dr. Ala is the recipient of the Junior Researcher Travel Grant, the Jordanian Royal Academic Sponsorship, and the Irbid Secondary Schools Outstanding Student Award.

Dr. Radu Stoleru is a Professor in the Department of Computer Science and Engineering at Texas A&M University, and heading the Laboratory for Embedded & Networked Sensor Systems (LENSS). His research interests are in deeply embedded wireless sensor systems, distributed systems, embedded computing, and computer networking. He is the recipient of the NSF CAREER Award in 2013. Dr. Stoleru received his Ph.D. in computer science from the University of Virginia in 2007. While at the University of Virginia, Dr. Stoleru received from the Department of Computer Science the Outstanding Graduate Student Research Award for 2007. He has authored or co-authored over 130 conference and journal papers with over 6,500 citations (Google Scholar). He is currently serving as an editorial board member for 3 international journals and has served as technical program committee member on numerous international conferences.



Dr. Guofei Gu is a professor and holder of the Eppright Professorship in Engineering in the Department of Computer Science & Engineering at Texas A&M University (TAMU). He received his Ph.D. degree in Computer Science from the College of Computing, Georgia Institute of Technology. His research interests are in network and systems security, such as malware and APT defense, software-defined programmable security, mobile and IoT security, and intrusion/anomaly detection. Dr. Gu is a recipient of 2010 NSF CAREER Award, 2013 AFOSR Young Investigator Award, 2010 IEEE S&P Best Student Paper Award, 2015 ICDCS Best Paper Award, Texas A&M Dean of Engineering Excellence Award, Presidential Impact Fellow, Charles H. Barclay Jr. 45 Faculty Fellow, TEES Faculty Fellow, TEES Research Impact Award, and Google Faculty Research Award. He has pioneered several new research directions such as botnet detection/defense and SDN security and served on the program committees of top-tier security conferences such as IEEE S&P, ACM CCS, USENIX Security, and NDSS. He is an IEEE Fellow and an ACM Distinguished Member. He is currently directing the SUCCESS Lab at TAMU.

Dr. Arnab Kumar Maity is an Associate Director in Pfizer, a renowned pharmaceutical company. His primary research interest is Bayesian Methods in Clinical Trials. In particular, he is working as a primary statistician of Phase I Cancer Studies. Besides, he is interested in the application of Variable Selection methods in high dimensional omics datasets and other settings. He did his post doctoral research in Texas A&M University and received Ph.D. from Northern Illinois University. Additionally, he is an associate editor of the Journal of Biopharmaceutical Statistics.

Dr. Suman Bhunia joined the Department of Computer Science and Software Engineering at the Miami University of Ohio as an assistant professor in Fall 2020. Previously, he was a postdoctoral researcher in the Department of Computer Science at Texas A&M University. Dr. Bhunia earned his Ph.D. degree in Computer Science from the University of Nevada, Reno, in 2017. Earlier, he obtained an M.Tech. in Distributed and Mobile Computing from Jadavpur University, Kolkata, in 2010 and a B.Tech in Electronics and Communication Engineering from West Bengal University of technology in 2008. Dr. Bhunia's research is focused on wireless communication and its security. At Miami, Dr. Bhunia is teaching courses such as Computer and Network Security, Ethical Hacking. He started a Cybersecurity student club at Miami.