CollusiveHijack: A New Route Hijacking Attack and Countermeasures in Opportunistic Networks

Ala Altaweel, Radu Stoleru, Guofei Gu
Department of Computer Science & Engineering
Texas A&M University
{altaweel, stoleru, guofei}@cse.tamu.edu

Arnab Kumar Maity
Department of Statistics
Texas A&M University
amaity@stat.tamu.edu

***Abstract*—In this paper, we first show that Hybrid Routing and Prophet protocols in Opportunistic Networks are vulnerable to the *CollusiveHijack* attack. In this attack, 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, with the goal of hijacking the routes of legitimate nodes. The CollusiveHijack enables Eve to launch more severe attacks like packet modification attack, traffic analysis attack, and incentive seeking attack. 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 two techniques to detect the CollusiveHijack attack: the Path Detection Technique (PDT) and the Hop Detection Technique (HDT), which trade off compatibility with the Bundle Security Protocol and the detection rate. We evaluated PDT and HDT through extensive simulations and a proof-of-concept system implementation. The results show that PDT and HDT are able to detect CollusiveHijack attacks with 80.0% and 99.4% detection rates, respectively (when Eve hijacks more than 60 packets) while maintaining a low false positive rate of 3.6%.**

I. Introduction

Opportunistic Mobile Networks (OMNs) refer to wireless networks in which mobile nodes are intermittently connected, without stable end-to-end paths. These networks have a wide range of applications, e.g., disaster response [1], battlefield communications, and social networks applications (Firechat and 1am [2] [3]). Due to the intermittent connectivity between the nodes in OMNs, their routing protocols (e.g., Hybrid Routing (HRP) [4] and Prophet Protocols [5] [6]) adopt the store-and-forward mechanism to deliver the packets. HRP and Prophet learn from the nodes' past contact history in order to make forwarding decisions. A pair of nodes measures the past contact frequencies to find the probability of future contacts. The Inter Contact Time (ICT) (i.e., 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 [7] [8].

One major security concern and research challenge for HRP and Prophet protocols is their vulnerability to a Route Hijacking attack. In this attack, 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 the OMN. The attack, called *CollusiveHijack*, 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 (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 [9], which enables Eve to corrupt the contents of the packets, thus enforcing packet retransmissions and a decrease in the packet delivery ratio [10] (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 [11] [12]; c) incentives seeking attack (i.e., if an incentive based mechanism [13] is employed in the OMN, 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 flood, wormhole, and packet dropping attacks [14]–[18]. However, many approaches have been proposed to detect route hijacking in Internet. Approaches [19]–[23] collect BGP updates and routing tables from a public BGP monitoring infrastructure [24]–[26] and raise alarms when a change in the origin Autonomous System (AS) of a prefix or a suspicious route is observed. Approaches [19]–[23] require network administrators (not available in OMNs) and need to create a list of owned/reached IPs a priori (the OMNs' nodes do not know their future contacts). Other approaches [27] [28] continuously probe Internet to detect whether any data path changes. They use pings/traceroutes 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. Due to the intermittent connectivity among the nodes in OMNs, pings/traceroutes fail to work. As a result, the CollusiveHijack attack is still an open research problem for OMNs.

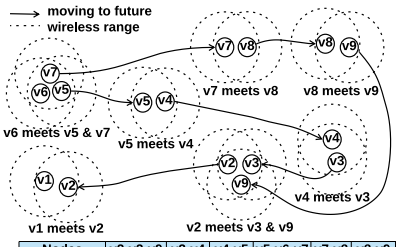
To address the aforementioned research challenge, we propose to detect the CollusiveHijack attack by employing the Kolmogorov-Smirnov two-sample test (KS2ST). The KS2ST [29] is a well known test for comparing two statistical distributions. That is, it measures the distance between the empirical distribution 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 [30]–[33]. We design two techniques to

detect the CollusiveHijack attack: the Path Detection Technique (PDT) and the Hop Detection Technique (HDT). PDT and HDT offer a trade-off between the *compatibility* with the Bundle Security Protocol (BSP) [34] (if additional steps are required at the intermediate nodes) and the *detection rate* 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 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 by Eve or not. HDT can achieve a higher detection rate against the CollusiveHijack attack than PDT.

The contributions of this paper are as follows: a) it demonstrates, via implementation and extensive simulations, a successful CollusiveHijack attack against HRP and Prophet protocols; b) it presents two new techniques, the Path Detection Technique (PDT) and the Hop Detection Technique (HDT), to detect the CollusiveHijack attack; c) it demonstrates the feasibility of PDT and HDT through extensive simulations and a proof-of-concept system implementation on Asus Eee notebooks; d) it demonstrates the effectiveness of PDT and HDT by showing that they identify CollusiveHijack attacks with a high detection rate while maintaining a low false positive rate.

II. Network and Adversary Models

We assume an OMN with nodes that run HRP or Prophet protocol. As shown in Figure 1, 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 (the scale of one second 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 claim	1	1	1	1	1	1
False claim	1	1	1	1	2	2

Fig. 1: Contact frequency for 1 day

1) **Hybrid Routing Protocol (HRP)** [4] [35] 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 [35].

2) **Prophet protocol** [5] [6] 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 $\Psi_{(v_6,v_5)} = \Psi_{(v_6,v_5)_{old}} \times \gamma^k$. $\gamma \in$ [0, 1) is a constant to decide how large impact the aging should have on Ψ and k 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 [36] [37].

Our adversary model assumes that the nodes have access to a public-key authentication service that is based on identity-based cryptography. Each node's private key is only known by itself to guarantee the authentication and message integrity. There are two types of nodes: honest nodes and *compromised* nodes. We assume that all nodes' users are benign. However, a malicious attacker, Eve, has control of some nodes that are infected by malware. We assume that Eve is not interested in launching a packet dropping or jamming attacks. However, Eve's nodes share each other's private keys and lie about their ICTs. For example, as shown 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. 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 because 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 ICT of v_8 and v_9 resets k to 0 and increases $\Psi_{(v_9,v_8)}$ due to the aging property. As a result of the transitive property ($\Psi_{(v_9,v_6)}$ = $\Psi_{(v_9,v_6)_{old}}$ + (1 - $\Psi_{(v_9,v_6)_{old}}$) $\times$ $\Psi_{(v_9,v_8)}$ $\times$ $\Psi_{(v_8,v_6)}$ $\times$ β) increases as well. This makes $\Psi_{(v_9,v_6)} > \Psi_{(v_2,v_6)}$ and results in forwarding v_1's packets to v_8 and v_9.

III. Motivation: CollusiveHijack against HRP and Prophet

We deploy the HRP and Prophet's daemon services [35] [36] on 11 Asus Eee notebooks that run Ubuntu 14.04 LTS. The wireless cards of the notebooks are set to operate in 2.4 GHz (channel 3) and in ad-hoc mode. 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.

For each experiment, we considered a normal scenario and an attack scenario. During the normal scenario, all nodes are honest about their ICTs. Differently, 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 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

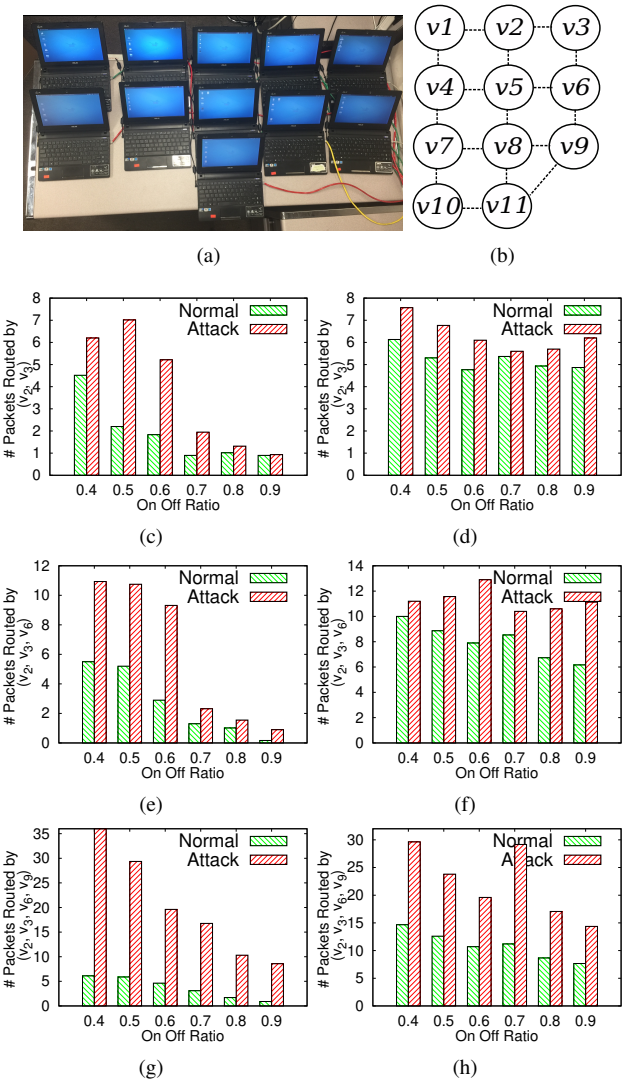


Fig. 2: (a) Attack implementation testbed. (b) Network topology. Routed packets for HRP (c, e, g) and Prophet (d, f, h).

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 [4] [35]. 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 can 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.

We also examined the impact of CollusiveHijack through extensive simulations in the ONE simulator [38]. We conducted trace-driven simulations using real world mobility trace, Reality [39], that consists of 97 users from MIT students, faculty and staff members. The mobility trace contains Bluetooth connection events over a time period of 9 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 [6] 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 varying the *lying factor* from 2 to 8. The lying factor reflects the ratio that the compromised nodes fake their ICTs by ($ICT_{faked} = \frac{ICT_{original}}{lying\ factor}$). We repeated each experiment 100 times and we averaged these 100 runs per each experiment. Figure 4(a) shows the total number of hijacked packets for two compromised nodes, which increases when the compromised nodes increase their lying factor. Also, increasing the number of the compromised nodes increases the total number of hijacked packets, as shown in Figures 4(b) and 4(c) for three and four compromised nodes, respectively.

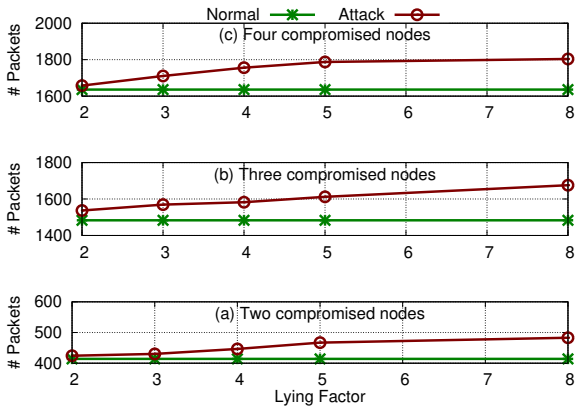


Fig. 4: Routed packets for Prophet

IV. CollusiveHijack Detection

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

A. 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 3. 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 3. 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 $P[D_{i,j} \leq d] = \frac{1}{\mathbb{E}[ICT_{i,j}]} \int_0^d (1 - P[ICT_{i,j} \leq z])dz$ and $\mathbb{E}[D_{i,j}] = \frac{\mathbb{E}[ICT_{i,j}]}{2} + \frac{\sigma^2(ICT_{i,j})}{2\mathbb{E}[ICT_{i,j}]}$ [7]. Where $\sigma^2(ICT_{i,j})$ is the variance of $ICT_{i,j}$. Previous research [4] [7] [8] 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 trace [39], and found that the pdfs of all ICTs among the users in this trace follow the exponential distribution. Accordingly, $\mathbb{E}[D_{i,j}] = \mathbb{E}[ICT_{i,j}]$. Let $P_{v_i,v_j} = \{v_i, v_1, v_2, .., v_{\eta-1}, v_j\}$ 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, which are independently and exponentially distributed. The pdf of P_{v_i,v_j} delay = $\lambda_{i,1}e^{-\lambda_{i,1}t} + \lambda_{1,2}e^{-\lambda_{1,2}t} + .. + \lambda_{\eta-1,j}e^{-\lambda_{\eta-1,j}t}$ is hypo-exponentially distributed $\sim Hypo(\lambda_{i,1}, \lambda_{1,2}, .., \lambda_{\eta-1,j})$ with mean = $1/[\lambda_{i,1} + \lambda_{1,2} + .. + \lambda_{\eta-1,j}]$ and variance = $1/[\lambda_{i,1}^2 + \lambda_{1,2}^2 + .. + \lambda_{\eta-1,j}^2]$ [40].

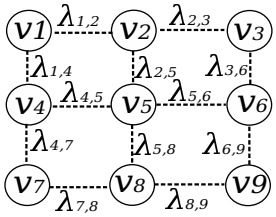


Fig. 3: OMN contact graph

The KS2ST is a well known non-parametric goodness-of-fit test [29]. This test measures the distance between the empirical cumulative distribution functions (CDFs) of two samples **a** = $\{a\}_{i=1}^n$ and **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 sample 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 α = 0.05. Accordingly, we reject the null hypothesis if the $p-value$ [41] of the KS2ST is < 0.05.

B. 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 that we present in Section IV-C.* 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. Moreover, the sender signs the inserted PCT and $Sender\_Id$, as shown in 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). In case a node receives a packet to be routed, it verifies the signed Id of the previous hop using its public key, as shown in line 10 of Algorithm 1. When the next hop

Algorithm 1 Steps at each node for PDT and HDT

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

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 honest

is available for the carrier node, it inserts $Carrier\_Id$ into the packet header. The carrier node also signs the inserted $Carrier\_Id$ before forwarding the packet (lines 12 and 15 of Algorithm 1). The aforementioned steps can be accomplished by including the nodes' payload integrity blocks in the packets (pages 24-25 of BSP [34]).

PDT is a detection-based protocol that runs by the destination nodes in OMNs. The pseudo code of PDT is presented in Algorithm 2. Once the destination receives a packet, it calculates the packet's delay and finds its path. Then, the destination verifies the inserted Ids into the packet header, using the sender and carriers' 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., $\lambda_{i,1}, \lambda_{1,2}, .., \lambda_{\eta-1,j}$). Notice that these contact frequencies are used by HRP and Prophet to make replication decisions. HRP and Prophet require these contact frequencies to be announced to all nodes in the OMN. Then, v_j employs the KS2ST to determine whether the delays of the received packets match the summation of the announced links' delays of P_{v_i,v_j} (i.e., the path delay of $P_{v_i,v_j} \sim Hypo(\lambda_{i,1}, \lambda_{1,2}, .., \lambda_{\eta-1,j})$. In the following paragraph, we present the steps of PDT through an example.

As an example, we assume that v_1 in Figure 3 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 aforementioned 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 the hypo-exponential distribution, $Hypo(\lambda_{1,2}, \lambda_{2,3}, \lambda_{3,6}, \lambda_{6,9})$, and performs a KS2ST between the 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, which 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 an honest path (lines 17-20 of Algorithm 2).

C. Hop Detection Technique (HDT)

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 [34], 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), then 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 IV-B (node v_1, in Figure 3, 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 inserted Ids, PCT, and PRT's of v_1, v_2, v_3, and v_6 using their public keys, as shown in 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 $[k \times (\eta + 1)]$

Algorithm 3 HDT at the destination

1: **for each** received $Packet$ via a $Path$ with $\boldsymbol{\eta}$ 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 $\boldsymbol{\eta}$ hops **do**
8: $PRT_{nodes} \leftarrow$

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}

(k rows; $\eta+1$ columns)
9: **for each** col_i in PRT_{nodes} **do**
10: $D_{hops}[col_i]$ = $PRT_{nodes}[col_{i+1}]$ - $PRT_{nodes}[col_i]$
11: $D_{hops} =$

$D_{1,(v_i,v_1)}$	$D_{1,(v_1,v_2)}$	..	$D_{1,(v_{\eta-1},v_j)}$
$D_{2,(v_i,v_1)}$	$D_{2,(v_1,v_2)}$	..	$D_{2,(v_{\eta-1},v_j)}$
..	..	..	..
$D_{k,(v_i,v_1)}$	$D_{k,(v_1,v_2)}$	..	$D_{k,(v_{\eta-1},v_j)}$

(k rows; η columns)
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

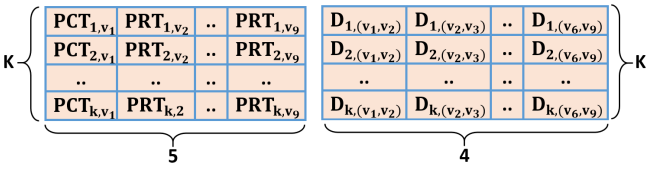


Fig. 5: PRT_{nodes} and D_{hops} built by v_9

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 decrementing 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 $D_{x,(y,w)}$ represents the delay of the x_{th} packet at $\textcircled{y} \longleftrightarrow \textcircled{w}$ 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 intermediate hops, $[(v_1, v_2), (v_2, v_3), (v_3, v_6), (v_6, v_9)]$, as a compromised or honest hop (lines 12-24 of Algorithm 3).

V. Implementation and Experimental Setup

We implemented PDT and HDT using R statistical language [42] on Asus Eee notebooks, that are 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 by the C++ user-space daemon services of HRP and Prophet protocols [35] [36] using Rcpp package [43]. We also enabled BSP [34] for HRP and Prophet daemon services. For the cryptographic sign and verify operations, we leveraged the Pairing-Based Cryptography (PBC) library [44] that implements the Hess identity-based signatures [45]. The PBC implementation uses 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 III in the ONE simulator [38]. We aim to investigate the performance of PDT and HDT 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 III) 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 two experiments against HRP on the testbed shown in Figure 2(a) for 0.4 and 0.9 on-off ratios. 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 [35] to collect the ICTs among the nodes. The HRP implementation leverages Optimized Link State Routing (OLSR) [46] for topology maintenance by invoking *olsrd* [47], an OLSR implementation, and fetches topology information from it. The *hrptclient* [35] measures the packet's delay at the destination. We evaluated our PDT and HDT in two aspects:

1) **Quantifying the overhead** of PDT and HDT 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** of the sign and verify operations.

2) **Evaluating the performance** of PDT and HDT using the following metrics: **a) Detection Rate:** the ability of PDT and HDT 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.

VI. Performance Evaluation

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

A. Quantifying the Overhead of PDT and HDT

In PDT and HDT, 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 HDT, each intermediate node adds its PRTs (4B). The signature field of the Hess identity-based scheme is 64B. Hence, the PDT and HDT's packet size overhead in bytes for $P_{v_i,v_j} = \{v_i, v_1, v_2, .., v_{\eta-1}, v_j\}$ path are $[68 \times \eta] + 4$ and $72 \times \eta$, respectively. For 10-hop path with an IPv4 packet size (64KB), the packet size overheads for PDT and HDT are $\sim$1.04% and $\sim$1.09%, respectively, which are relatively small. In order to calculate the execution time of the sign and verify operations of the Hess identity-based scheme [45], we averaged 10,000 measurements of these operations on one Asus Eee notebooks. The execution time of the sign and verify operations are 118 msec and 42 msec, respectively.

B. Performance Evaluation of PDT and HDT

Evaluation in the ONE simulator. We conducted the first experiment on a path with η = 6 hops including one compromised hop. We performed the following steps. First, we extracted 1,000 different paths with 6 hops from the trace-driven 100 experiments that we conducted in Section III. 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 Figure 4(a), that it is attractive for the compromised nodes to increase their lying factor to hijack more packets (the total hijacked packets are 10 and 80 for 2 and 8 lying factors, respectively). However, increasing the number of hijacked packets enhances the PDT's detection 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 performs a hop-wise detection against the CollusiveHijack attack, we present its detection rate on 1-hop for different k values in Figure 6(h). As it is clear in Figure 6(h), HDT's detection rates are $> 98\%$ for all lying factors when $k > 40$. Hence, we recommend to leverage HDT to effectively detect the CollusiveHijack attack despite it's overhead and incompatibility with BSP [34]. The false positive rates of HDT (i.e., on 1-hop) are relatively small, as shown in Figure 7(a).

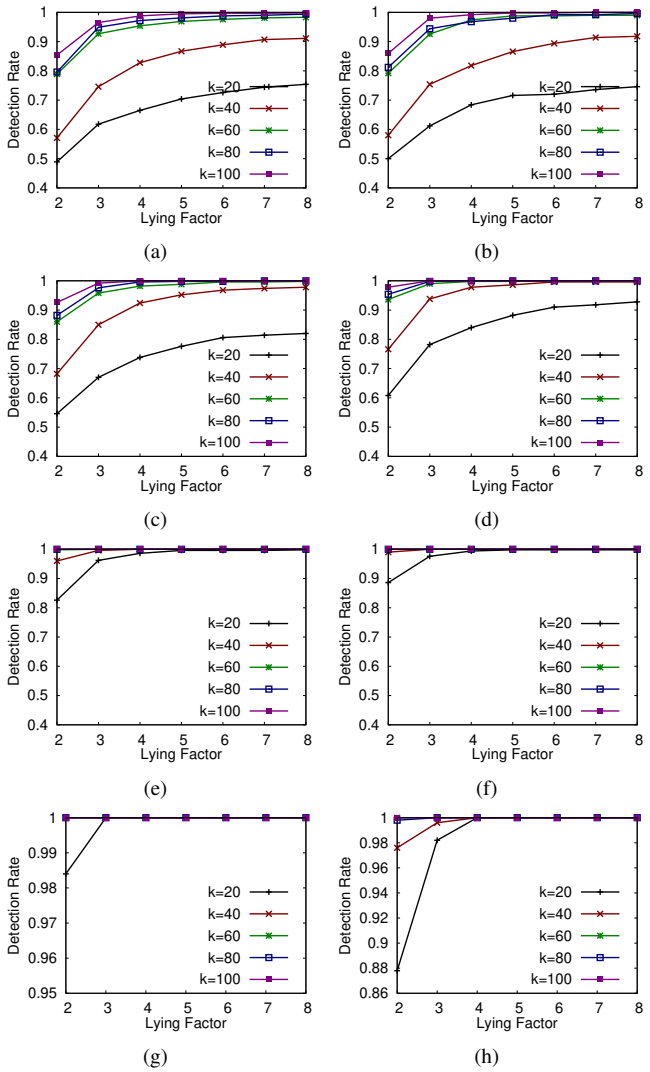


Fig. 6: PDT's Detection Rate when the ratio of the compromised links: (a) 16.6%($\frac{1}{6}$). (b) 20%($\frac{1}{5}$). (c) 25%($\frac{1}{4}$). (d) 33%($\frac{2}{6}$). (e) 40%($\frac{2}{5}$). (f) 50%($\frac{3}{6}$). (g) 60%($\frac{3}{5}$). (h) HDT's Detection Rate.

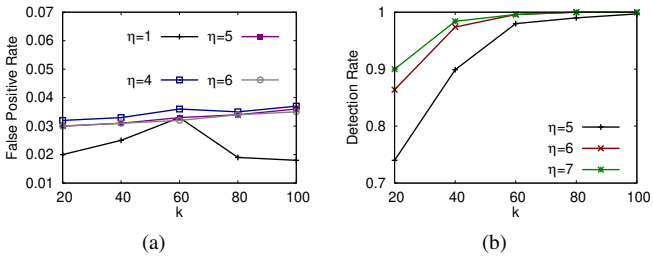


Fig. 7: (a) False Positive Rate of PDT (η = 4, 5, 6 hops) and HDT (η = 1 hop). (b) PDT Detection Rate Against Fake Ids Attack.

System evaluation on Asus notebooks testbed. Figures 8(a)

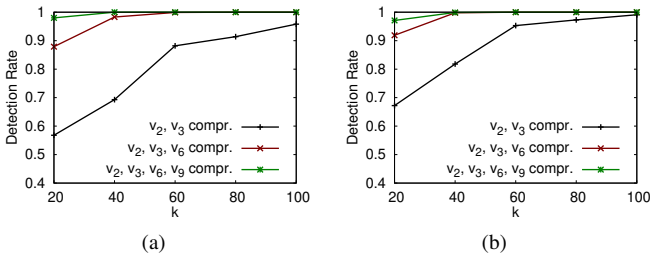


Fig. 8: PDT's Detection Rate for on-off ratio = (a) 0.4 (b) 0.9

and 8(b) show the detection rate of PDT for 0.4 and 0.9 on-off ratios. 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 investigated the HDT detection capability against the CollusiveHijack attack for these experiments and found that its detection rate is $> 98.0\%$ when $k > 40$ (omitted here due to space constraints). The false positive rates of PDT and HDT for these experiments are $< 3.5\%$ (close to the rates shown in Figure 7(a)).

VII. Security Analysis

Based on the attacker model described in Section II, Eve can launch the following types of attacks:

1) Fake Ids attack against PDT: Eve might pretend that two or more of her nodes have the same Id to not be detected by PDT. Without loss of generality, if 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 VI-B) for η = 5, 6, and 7 hops. 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. The PDT detection rate is $> 98.0\%$ when $k > 60$, as shown in Figure 7(b). Hence, if Eve launches the fake Ids attack, one of her nodes will be detected (v_2 in $P_{s,d}$ above). We also calculated the detection rates against fake Ids attack for 3 compromised nodes (all have same Id) and found that it is $> 99.0\%$ when $k > 40$ (the results are omitted here due to space constraints).

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

Without loss of generality, if Eve compromises v_2 and v_3 and fakes their contact frequencies to $2 \times \lambda_{v_2,v_3}$, she also can fake the PRT at v_3 to $\frac{\tau_3}{2}$ to not be detected by HDT 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, our HDT is able to detect the source node of $(v_3) \xleftrightarrow[\frac{\tau_3}{2}+\tau_4]{\lambda_{v_3,v_4}} (v_4)$, as shown in line 22 of Algorithm 3 (i.e., since 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, $v_1, v_2, .., v_n$ in a path, she can launch fake PRTs attack by faking the PRTs among $v_1, v_2, .., v_{n-1}$ nodes, however, the last compromised node, v_n, will be detected by HDT.

VIII. State of the Art

Most 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* [19]–[23] collect BGP updates or routing tables from a distributed set of public BGP monitoring infrastructure and route collectors such as [24]–[26], and raise alarms when a change in the origin-AS of a prefix, or a suspicious route is observed. PHAS and Cyclops [21] [22] are notification systems that alert prefix owners (i.e., ISPs) when their BGP origin change. The network administrator in ARTEMIS [23] 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. [24]–[26]). 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* [27] [28] 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 [27] of a prefix or the paths leading to it [28] are observed. The Listen protocol [48] 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* [49]–[51] combine control and data plane information to detect the hijacking attack, however, they cannot detect the CollusiveHijack attack because they still need network administrators [51]

and use monitor tools like pings and traceroutes [50] [49]. *Cryptographic approaches* [52]–[54] 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 [54] and Whisper protocol [48] 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 [52].

IX. Conclusions and Future Work

We presented two novel algorithms, PDT and HDT, to detect the CollusiveHijack attack in OMNs. Both PDT and HDT employ the KS2ST and offer a trade-off between the compatibility with BSP and the detection rate against the CollusiveHijack attack. Our evaluations show that both algorithms are able to identify the CollusiveHijack attack with $\sim$80% and 99.4% detection rates, respectively, and $\sim$3.6% false positive rate. In our future work, we plan to implement both PDT and HDT in smartphones and validate their detection capabilities with real world OMNs' users.

Acknowledgment

This material is based upon a work supported by National Institute of Standards and Technology (NIST) under grant NO. (#70NANB17H190). We appreciate the helpful suggestions from the reviewers. Also, we thank Suman Bhunia and Mengyuan Chao for helpful discussions on this paper.

References

[1] Distressnet-ng project. http://distressnet.net/.
[2] Firechat: Google play store. https://play.google.com/store.
[3] 1am: Censorship-resistant microblogging. http://1am-networks.org.
[4] C. Yang and R. Stoleru. Hybrid routing in wireless networks with diverse connectivity. *MobiHoc'16*.
[5] A. Lindgren, A. Doria, E. Davies, and O. Schelén. Probabilistic routing in intermittently connected networks. *SIGMOBILE Mob. Comput. Commun. Rev.*, 7(3), 2003.
[6] A. Lindgren, A. Doria, E. Davies, and S. Grasic. Probabilistic routing protocol for intermittently connected networks. irtf rfc 6693. 2012.
[7] X. Tie, A. Venkataramani, and A. Balasubramanian. R3: Robust replication routing in wireless networks with diverse connectivity characteristics. *MobiCom'11*.
[8] A. Balasubramanian, B. Levine, and A. Venkataramani. Dtn routing as a resource allocation problem. *SIGCOMM'07*.
[9] R. Patil and M. P. Tahiliani. Detecting packet modification attack by misbehaving router. *ICNSC'14*.
[10] J. Burgess, G. D. Bissias, M. Corner, and B. N. Levine. Surviving attacks on disruption-tolerant networks without authentication. *MobiHoc'07*.
[11] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic. Robust smartphone app identification via encrypted network traffic analysis. *IEEE TIFS*, 13(1), 2018.
[12] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic. Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic. *EuroS&P'16*.
[13] 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.
[14] 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.
[15] Y. Ren, M. C. Chuah, J. Yang, and Y. Chen. Detecting wormhole attacks in delay-tolerant networks. *IEEE Wireless Comm.*, 17(5), 2010.
[16] F. Li, J. Wu, and A. Srinivasan. Thwarting blackhole attacks in disruption-tolerant networks using encounter tickets. *INFOCOM'09*.
[17] Q. Li and G. Cao. Mitigating routing misbehavior in disruption tolerant networks. *IEEE TIFS*, 7(2), 2012.
[18] M. Alajeely, R. Doss, A. Ahmad, and V. Mak-Hau. Defense against packet collusion attacks in opportunistic networks. *Computers and Security*, 65, 2017.
[19] Bgpmon. http://www.bgpmon.net.
[20] Ripe myasn system. http://www.ris.ripe.net/myasn.html.
[21] M. Lad, D. Massey, D. Pei, Y. Wu, B. Zhang, and L. Zhang. Phas: A prefix hijack alert system. *USENIX'06*.
[22] Y. Chi, R. Oliveira, and L. Zhang. Cyclops: The as-level connectivity observatory. *ACM SIGCOMM Computer Comm. Review*, 38(5), 2008.
[23] 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*.
[24] Bgpmon (colorado state university). https://www.bgpmon.io/.
[25] Routing information service (ris). https://www.ripe.net/.
[26] Route views project(university of oregon). http://www.routeviews.org/.
[27] 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.
[28] 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*.
[29] J. Frank and Jr. Massey. The kolmogorov-smirnov test for goodness of fit. *Journal of the American Statistical Association*, 46(253), 1951.
[30] S. Gianvecchio and H. Wang. An entropy-based approach to detecting covert timing channels. *IEEE TDSC*, 8(6), 2011.
[31] A. L. Toledo and X. Wang. Robust detection of selfish misbehavior in wireless networks. *IEEE JSAC*, 25(6), 2007.
[32] J. Tapiador, P. Teodoro, and J. Verdejo. Measuring normality in http traffic for anomaly-based intrusion detection. *Computer Networks*, 45(2), 2004.
[33] J. Caberera, B. Ravichandran, and R. Mehra. Statistical traffic modeling for network intrusion detection. *MASCOTS'00*.
[34] S. Symington, S. Farrell, H. Weiss, and P. Lovell. Bundle security protocol specification. https://tools.ietf.org/html/rfc6257. 2011.
[35] C. Yang and R. Stoleru. Hybrid routing in wireless networks with diverse connectivity. *Technical Report, Texas A&M University*.
[36] Ibr-dtn - a modular and lightweight implementation of the bundle protocol. https://github.com/ibrdtn/.
[37] M. Doering, S. Lahde, J. Morgenroth, and L. Wolf. Ibr-dtn: An efficient implementation for embedded systems. *CHANTS'08*.
[38] A. Keränen, J. Ott, and T. Kärkkäinen. The one simulator for dtn protocol evaluation. *Simutools'09*.
[39] N. Eagle and A. Pentland. Reality mining: Sensing complex social systems. *Personal Ubiquitous Comput.*, 10(4), 2006.
[40] K. Smaili, T. Kadri, and S. Kadry. Hypoexponential distribution with different parameters. *Applied Mathematics*, 4(4), 2013.
[41] J. C. Ferreira and C. M. Patino. What does the p value really mean? *Jornal Brasileiro de Pneumologia*, 41(5), 2015.
[42] The r project for statistical computing. https://www.r-project.org/.
[43] Rcpp for seamless r and c++ integration. http://www.rcpp.org/.
[44] B. Lynn. The pairing-based cryptography (pbc) library. https://crypto.stanford.edu/pbc/.
[45] F. Hess. Efficient identity based signature schemes based on pairings. *SAC'02*.
[46] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot. Optimized link state routing protocol for ad hoc networks. *INMIC'01*.
[47] A. Tonnesen. Implementing and extending the optimized link state routing protocol. *Masters thesis, University of Oslo'04*.
[48] L. Subramanian, V. Roth, I. Stoica, S. Shenker, and R. H. Katz. Listen and whisper: Security mechanisms for bgp. *NSDI'04*.
[49] X. Hu and Z. M. Mao. Accurate real-time identification of ip prefix hijacking. *S&P'07*.
[50] X. Shi, Y. Xiang, Z. Wang, X. Yin, and J. Wu. Detecting prefix hijackings in the internet with argus. *IMC'12*.
[51] J. Schlamp, R. Holz, Q. Jacquemart, G. Carle, and E. W. Biersack. Heap: Reliable assessment of bgp hijacking attacks. *IEEE JSAC*, 34(6), 2016.
[52] M. Lepinski and K. Sriram. Bgpsec protocol specification. https://tools.ietf.org/html/rfc8205. 2017.
[53] M. Lepinski and S. Kent. An infrastructure to support secure internet routing. https://tools.ietf.org/html/rfc6480. 2012.
[54] S. Kent, C. Lynn, and K. Seo. Secure border gateway protocol (s-bgp). *IEEE JSAC*, 18(4), 2000.