To Lie or to Comply: Defending against Flood Attacks in Disruption Tolerant Networks

Qinghua Li, *Student Member*, *IEEE*, Wei Gao, *Member*, *IEEE*, Sencun Zhu, and Guohong Cao, *Fellow*, *IEEE*

Abstract—Disruption Tolerant Networks (DTNs) utilize the mobility of nodes and the opportunistic contacts among nodes for data communications. Due to the limitation in network resources such as contact opportunity and buffer space, DTNs are vulnerable to flood attacks in which attackers send as many packets or packet replicas as possible to the network, in order to deplete or overuse the limited network resources. In this paper, we employ rate limiting to defend against flood attacks in DTNs, such that each node has a limit over the number of packets that it can generate in each time interval and a limit over the number of replicas that it can generate for each packet. We propose a distributed scheme to detect if a node has violated its rate limits. To address the challenge that it is difficult to count all the packets or replicas sent by a node due to lack of communication infrastructure, our detection adopts *claim-carry-and-check*: each node itself counts the number of packets or replicas that it has sent and claims the count to other nodes; the receiving nodes carry the claims when they move, and cross-check if their carried claims are inconsistent when they contact. The claim structure uses the pigeonhole principle to guarantee that an attacker will make inconsistent claims which may lead to detection. We provide rigorous analysis on the probability of detection, and evaluate the effectiveness and efficiency of our scheme with extensive trace-driven simulations.

Index Terms—DTN, security, flood attack, detection

1 Introduction

Disruption Tolerant Networks (DTNs) [1] consist of mobile nodes carried by human beings [2], [3], vehicles [4], [5], etc. DTNs enable data transfer when mobile nodes are only intermittently connected, making them appropriate for applications where no communication infrastructure is available such as military scenarios and rural areas. Due to lack of consistent connectivity, two nodes can only exchange data when they move into the transmission range of each other (which is called a *contact* between them). DTNs employ such contact opportunity for data forwarding with "store-carry-and-forward"; i.e., when a node receives some packets, it stores these packets in its buffer, carries them around until it contacts another node, and then forwards them. Since the contacts between nodes are opportunistic and the duration of a contact may be short because of mobility, the usable bandwidth which is only available during the opportunistic contacts is a limited resource. Also, mobile nodes may have limited buffer space.

Due to the limitation in bandwidth and buffer space, DTNs are vulnerable to flood attacks. In flood attacks, maliciously or selfishly motivated attackers inject as many packets as possible into the network, or instead of injecting different packets the attackers forward replicas of the same packet to as many nodes as possible. For convenience, we call the two types of attack *packet flood attack* and *replica flood attack*, respectively. Flooded packets and replicas can waste the precious bandwidth and buffer resources, prevent benign packets from being forwarded and thus degrade the network service provided to good nodes. Moreover, mobile nodes spend much energy on transmitting/receiving flooded packets and replicas which may shorten their battery life. Therefore, it is urgent to secure DTNs against flood attacks.

Although many schemes have been proposed to defend against flood attacks on the Internet [6] and in wireless sensor networks [7], they assume persistent connectivity and cannot be directly applied to DTNs that have intermittent connectivity. In DTNs, little work has been done on flood attacks, despite the many works on routing [8], [4], [36], data dissemination [9], [37], blackhole attack [10], wormhole attack [11], and selfish dropping behavior [12], [13]. We noted that the packets flooded by outsider attackers (i.e., the attackers without valid cryptographic credentials) can be easily filtered with authentication techniques (e.g., [14]). However, authentication alone does not work when insider attackers (i.e., the attackers with valid cryptographic credentials) flood packets and replicas with valid signatures. Thus, it is still an open problem is to address flood attacks in DTNs.

In this paper, we employ rate limiting [15] to defend against flood attacks in DTNs. In our approach, each node has a limit over the number of packets that it, as a source node, can send to the network in each time interval. Each node also has a limit over the number of replicas that it can

- Q. Li and G. Cao are with the Department of Computer Science and Engineering, The Pennsylvania State University, IST Building, University Park, PA 16802. E-mail: {qxl118, gcao}@cse.psu.edu.
- W. Gao is with the Department of Electrical Engineering and Computer Science, The University of Tennessee, 302 Min Kao, 1520 Middle Drive, Knoxville, TN 37996. E-mail: weigao@utk.edu.
- S. Zhu is with the Department of Computer Science and Engineering, College of Information Sciences and Technology, The Pennsylvania State University, 338F IST Building, University Park, PA 16802. E-mail: szhu@cse.psu.edu.

Manuscript received 1 May 2011; revised 26 Mar. 2011; accepted 5 Oct. 2012; published online 19 Oct. 2012.
For information on obtaining reprints of this article, please send e-mail to: tdsc@computer.org, and reference IEEECS Log Number TDSC-2011-05-0125.
Digital Object Identifier no. 10.1109/TDSC.2012.84.

generate for each packet (i.e., the number of nodes that it can forward each packet to). The two limits are used to mitigate packet flood and replica flood attacks, respectively. If a node violates its rate limits, it will be detected and its data traffic will be filtered. In this way, the amount of flooded traffic can be controlled.

Our main contribution is a technique to detect if a node has violated its rate limits. Although it is easy to detect the violation of rate limit on the Internet and in telecommunication networks where the egress router and base station can account each user's traffic, it is challenging in DTNs due to lack of communication infrastructure and consistent connectivity. Since a node moves around and may send data to any contacted node, it is very difficult to count the number of packets or replicas sent out by this node. Our basic idea of detection is *claim-carry-and-check*. Each node *itself* counts the number of packets or replicas that it has sent out, and claims the count to other nodes; the receiving nodes carry the claims around when they move, exchange some claims when they contact, and cross-check if these claims are inconsistent. If an attacker floods more packets or replicas than its limit, it has to use the same count in more than one claim according to the pigeonhole principle,[1] and this inconsistency may lead to detection. Based on this idea, we use different cryptographic constructions to detect packet flood and replica flood attacks.

Because the contacts in DTNs are opportunistic in nature, our approach provides probabilistic detection. The more traffic an attacker floods, the more likely it will be detected. The detection probability can be flexibly adjusted by system parameters that control the amount of claims exchanged in a contact. We provide a lower and upper bound of detection probability and investigate the problem of parameter selection to maximize detection probability under a certain amount of exchanged claims. The effectiveness and efficiency of our scheme are evaluated with extensive trace-driven simulations.

This paper is structured as follows. Section 2 motivates our work. Section 3 presents our models and basic ideas. Sections 4 and 5 present our scheme. Section 6 presents security and cost analysis. Section 7 presents simulation results. The last two sections present related work and conclusions, respectively.

2 Motivation

2.1 The Potential Prevalence of Flood Attacks

Many nodes may launch flood attacks for malicious or selfish purposes. Malicious nodes, which can be the nodes deliberately deployed by the adversary or subverted by the adversary via mobile phone worms [16], launch attacks to congest the network and waste the resources of other nodes.

Selfish nodes may also exploit flood attacks to increase their communication throughput. In DTNs, a single packet usually can only be delivered to the destination with a probability smaller than 1 due to the opportunistic connectivity. If a selfish node floods many replicas of its own packet, it can increase the likelihood of its packet being delivered, since the delivery of any replica means successful delivery of the packet. With packet flood attacks, selfish nodes can also increase their throughput, albeit in a subtler manner. For example, suppose Alice wants to send a packet to Bob. Alice can construct 100 variants of the original packet which only differ in one unimportant padding byte, and send the 100 variants to Bob independently. When Bob receives any one of the 100 variants, he throws away the padding byte and gets the original packet.

1. The pigeonhole principle states that if α items are put into β pigeonholes with $\alpha > \beta$, then at least one pigeonhole must contain more than one item.

2.2 The Effect of Flood Attacks

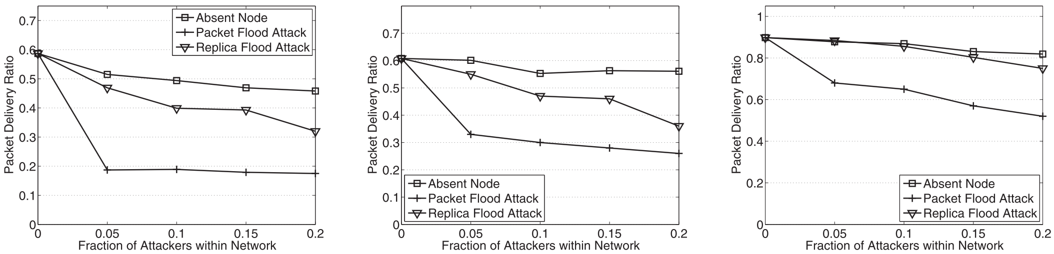
To study the effect of flood attacks on DTN routing and motivate our work, we run simulations on the MIT Reality trace [17] (see more details about this trace in Section 7).

We consider three general routing strategies in DTNs. 1) *Single-copy routing* (e.g., [18], [8]): after forwarding a packet out, a node deletes its own copy of the packet. Thus, each packet only has one copy in the network. 2) *Multicopy routing* (e.g., [19]): the source node of a packet sprays a certain number of copies of the packet to other nodes and each copy is individually routed using the single-copy strategy. The maximum number of copies that each packet can have is fixed. 3) *Propagation routing* (e.g., [17], [20], [21]): when a node finds it appropriate (according to the routing algorithm) to forward a packet to another encountered node, it replicates that packet to the encountered node and keeps its own copy. There is no preset limit over the number of copies a packet can have. In our simulations, SimBet [8], Spray-and-Focus [19] (three copies allowed for each packet) and Propagation are used as representatives of the three routing strategies, respectively. In Propagation, a node replicates a packet to another encountered node if the latter has more frequent contacts with the destination of the packet.

Two metrics are used, The first metric is packet delivery ratio, which is defined as the fraction of packets delivered to their destinations out of all the unique packets generated. The second metric is the fraction of wasted transmissions (i.e., the transmissions made by good nodes for flooded packets). The higher fraction of wasted transmissions, the more network resources are wasted. We noticed that the effect of packet flood attacks on packet delivery ratio has been studied by Burgess et al. [22] using a different trace [4]. Their simulations show that packet flood attacks significantly reduce the packet delivery ratio of single-copy routing but do not affect propagation routing much. However, they do not study replica flood attacks and the effect of packet flood attacks on wasted transmissions.

In our simulations, a packet flood attacker floods packets destined to random good nodes in each contact until the contact ends or the contacted node's buffer is full. A replica flood attacker replicates the packets it has generated to every encountered node that does not have a copy. Each good node generates thirty packets on the 121st day of the Reality trace, and each attacker does the same in replica flood attacks. Each packet expires in 60 days. The buffer size of each node is 5 MB, bandwidth is 2 Mbps and packet size is 10 KB.

Fig. 1 shows the effect of flood attacks on packet delivery ratio. Packet flood attack can dramatically reduce the packet delivery ratio of all three types of routing. When the fraction of attackers is high, replica flood attack can significantly decrease the packet delivery ratio of single-copy and multicopy routing, but it does not have much effect on propagation routing.



(a) Single-copy Routing (b) Multi-copy Routing (c) Propagation Routing

Fig. 1. The effect of flood attacks on packet delivery ratio. In absent node, attackers are simply removed from the network. Attackers are selectively deployed to high-connectivity nodes.

Fig. 2 shows the effect of flood attacks on wasted transmission. Packet flood attack can waste more than 80 percent of the transmissions made by good nodes in all routing strategies when the fraction of attackers is higher than 5 percent. When 20 percent of nodes are attackers, replica flood attack can waste 68 and 44 percent of transmissions in single-copy and multicopy routing, respectively. However, replica flood attack only wastes 17 percent of transmissions in propagation routing. This is because each good packet is also replicated many times.

Remarks. The results show that all the three types of routing are vulnerable to packet flood attack. Single-copy and multicopy routing are also vulnerable to replica flood attack, but propagation routing is much more resistant to replica flood. *Motivated by these results, this paper addresses packet flood attack without assuming any specific routing strategy, and addresses replica flood attack for single-copy and multicopy routing only.*

3 Overview

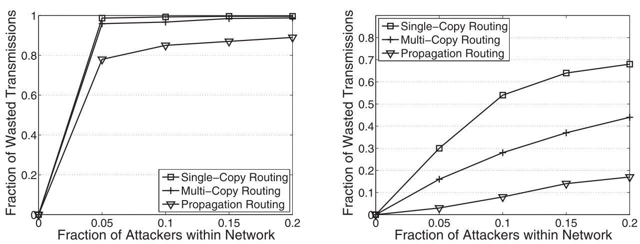
3.1 Problem Definition

3.1.1 Defense against Packet Flood Attacks

We consider a scenario where each node has a rate limit L on the number of unique packets that it as a source can generate and send into the network within each time interval T. The time intervals start from time 0, T, $2T$, etc. The packets generated within the rate limit are deemed legitimate, but the packets generated beyond the limit are deemed flooded by this node. To defend against packet flood attacks, our goal is to detect if a node as a source has generated and sent more unique packets into the network than its rate limit L per time interval.

A node's rate limit L does not depend on any specific routing protocol, but it can be determined by a service contract between the node and the network operator as discussed in Section 3.1.3. Different nodes can have different rate limits and their rate limits can be dynamically adjusted.

The length of time interval should be set appropriately. If the interval is too long, rate limiting may not be very effective against packet flood attacks. If the interval is too short, the number of contacts that each node has during one interval may be too nondeterministic and thus it is difficult to set an appropriate rate limit. Generally speaking, the interval should be short under the condition that most nodes can have a significant number of contacts with other nodes within one interval, but the appropriate length depends on the contact patterns between nodes in the specific deployment scenario.



(a) Packet Flood Attack (b) Replica Flood Attack

Fig. 2. The effect of flood attacks on the fraction of wasted transmission. Attackers are randomly deployed.

3.1.2 Defense against Replica Flood Attacks

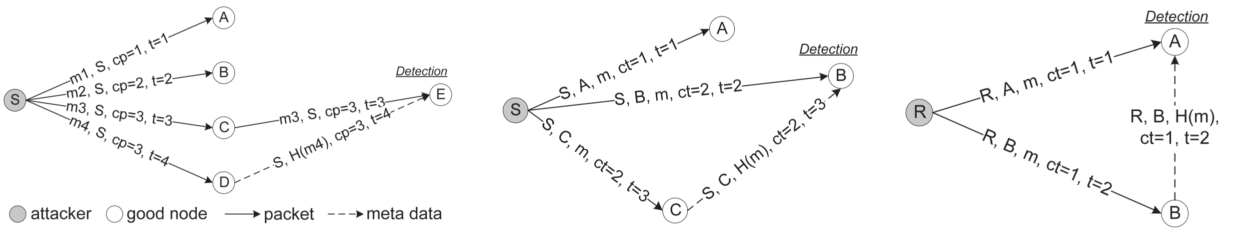
As motivated in Section 2, the defense against replica flood considers single-copy and multicopy routing protocols. These protocols require that, for each packet that a node buffers no matter if this packet has been generated by the node or forwarded to it, there is a limit l on the number of times that the node can forward this packet to other nodes. The values of l may be different for different buffered packets. Our goal is to detect if a node has violated the routing protocol and forwarded a packet more times than its limit l for the packet.

A node's limit l for a buffered packet is determined by the routing protocol. In multicopy routing, $l = L'$ (where L' is a parameter of routing) if the node is the source of the packet, and $l = 1$ if the node is an intermediate hop (i.e., it received the packet from another node). In single-copy routing, $l = 1$ no matter if the node is the source or an intermediate hop. Note that the two limits L and l do not depend on each other.

We discuss how to defend against replica flood attacks for quota-based routing [23], [19], [24] in Section 4.9.

3.1.3 Setting the Rate Limit L

One possible method is to set L in a request-approve style. When a user joins the network, she requests for a rate limit from a trusted authority which acts as the network operator. In the request, this user specifies an appropriate value of L based on prediction of her traffic demand. If the trusted authority approves this request, it issues a *rate limit certificate* to this user, which can be used by the user to prove to other nodes the legitimacy of her rate limit. To prevent users from requesting unreasonably large rate limits, a user pays an appropriate amount of money or



(a) Packet flood ($L = 3$) (b) Replica flood by the source ($l = 2$) (c) Replica flood by a relay ($l = 1$)

Fig. 3. The basic idea of flood attack detection. cp and ct are packet count and transmission count, respectively. The arrows mean the transmission of packet or metadata which happens when the two end nodes contact.

virtual currency (e.g., the credits that she earns by forwarding data for other users [25]) for her rate limit. When a user predicts an increase (decrease) of her demand, she can request for a higher (lower) rate limit. The request and approval of rate limit may be done offline. The flexibility of rate limit leaves legitimate users' usage of the network unhindered. This process can be similar to signing a contract between a smartphone user and a 3G service provider: the user selects a data plan (e.g., 200 MB/month) and pays for it; she can upgrade or downgrade the plan when needed.

3.2 Models and Assumptions

3.2.1 Network Model

In DTNs, since contact durations may be short, a large data item is usually split into smaller packets (or fragments) to facilitate data transfer. For simplicity, we assume that all packets have the same predefined size. Although in DTNs the allowed delay of packet delivery is usually long, it is still impractical to allow unlimited delays. Thus, we assume that each packet has a lifetime. The packet becomes meaningless after its lifetime ends and will be discarded.

We assume that every packet generated by nodes is unique. This can be implemented by including the source node ID and a locally unique sequence number, which is assigned by the source for this packet, in the packet header.

We also assume that time is loosely synchronized, such that any two nodes are in the same time slot at any time. Since the intercontact time in DTNs is usually at the scale of minutes or hours, the time slot can be at the scale of one minute. Such loose time synchronization is not hard to achieve.

3.2.2 Adversary Model

There are a number of attackers in the network. An attacker can flood packets and/or replicas. When flooding packets, the attacker acts as a source node. It creates and injects more packets into the network than its rate limit L. When flooding replicas, the attacker forwards its buffered packets (which can be generated by itself or received from other nodes) more times than its limit l for them. The attackers may be insiders with valid cryptographic keys. Some attackers may collude and communicate via out-band channels.

3.2.3 Trust Model

We assume that a public-key cryptography system is available. For example, Identity-Based Cryptography (IBC) [26] has been shown to be practical for DTNs [27]. In IBC, only an offline Key Generation Center (KGC) is needed. KGC generates a private key for each node based on the node's id, and publishes a small set of public security parameters to the node. Except the KGC, no party can generate the private key for a node id. With such a system, an attacker cannot forge a node id and private key pair. Also, attackers do not know the private key of a good node (not attacker).

Each node has a *rate limit certificate* obtained from a trusted authority. The certificate includes the node's ID, its approved rate limit L, the validation time of this certificate and the trusted authority's signature. The rate limit certificate can be merged into the public key certificate or stand alone.

3.3 Basic Idea: Claim-Carry-and-Check

3.3.1 Packet Flood Detection

To detect the attackers that violate their rate limit L, we must count the number of unique packets that each node as a source has generated and sent to the network in the current interval. However, since the node may send its packets to any node it contacts at any time and place, no other node can monitor all of its sending activities. To address this challenge, our idea is to let the node itself count the number of unique packets that it, as a source, has sent out, and *claim* the up-to-date packet count (together with a little auxiliary information such as its ID and a timestamp) in each packet sent out. The node's rate limit certificate is also attached to the packet, such that other nodes receiving the packet can learn its authorized rate limit L. If an attacker is flooding more packets than its rate limit, it has to dishonestly claim a count smaller than the real value in the flooded packet, since the real value is larger than its rate limit and thus a clear indicator of attack. The claimed count must have been used before by the attacker in another claim, which is guaranteed by the pigeonhole principle, and these two claims are inconsistent. The nodes which have received packets from the attacker *carry* the claims included in those packets when they move around. When two of them contact, they *check* if there is any inconsistency between their collected claims. The attacker is detected when an inconsistency is found.

Let us look at an example in Fig. 3a. S is an attacker that successively sends out four packets to A, B, C, and D, respectively. Since $L = 3$, if S claims the true count 4 in the fourth packet $m4$, this packet will be discarded by D. Thus, S dishonestly claims the count to be 3, which has already been claimed in the third packet $m3$. $m3$ (including the claim) is further forwarded to node E. When D and E

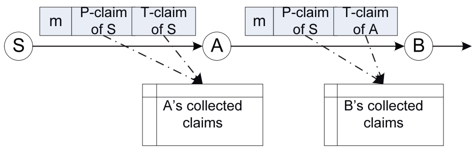


Fig. 4. The conceptual structure of a packet and the changes made at each hop of the forwarding path.

contact, they exchange the count claims included in $m3$ and $m4$, and check that S has used the same count value in two different packets. Thus, they detect that S as an attacker.

3.3.2 Replica Flood Detection

Claim-carry-and-check can also be used to detect the attacker that forwards a buffered packet more times than its limit l. Specifically, when the source node of a packet or an intermediate hop transmits the packet to its next hop, it claims a transmission count which means the number of times it has transmitted this packet (including the current transmission). Based on if the node is the source or an intermediate node and which routing protocol is used, the next hop can know the node's limit l for the packet, and ensure that the claimed count is within the correct range $[1, l]$. Thus, if an attacker wants to transmit the packet more than l times, it must claim a false count which has been used before. Similarly as in packet flood attacks, the attacker can be detected. Examples are given in Figs. 3b and 3c.

4 Our Scheme

Our scheme uses two different cryptographic constructions to detect packet flood and replica flood attacks independently. When our scheme is deployed to propagation routing protocols, the detection of replica flood attacks is deactivated.

The detection of packet flood attacks works independently for each time interval. Without loss of generality, we only consider one time interval when describing our scheme. For convenience, we first describe our scheme assuming that all nodes have the same rate limit L, and relax this assumption in Section 4.8. In the following, we use $SIG_i(*)$ to denote node i's signature over the content in the brackets.

4.1 Claim Construction

Two pieces of metadata are added to each packet (see Fig. 4), Packet Count Claim (P-claim) and Transmission Count Claim (T-claim). P-claim and T-claim are used to detect packet flood and replica flood attacks, respectively.

P-claim is added by the source and transmitted to later hops along with the packet. T-claim is generated and processed hop-by-hop. Specifically, the source generates a T-claim and appends it to the packet. When the first hop receives this packet, it peels off the T-claim; when it forwards the packet out, it appends a new T-claim to the packet. This process continues in later hops. Each hop keeps the P-claim of the source and the T-claim of its previous hop to detect attacks.

4.1.1 P-Claim

When a source node S sends a new packet m (which has been generated by S and not sent out before) to a contacted node, it generates a P-claim as follows:

$$\text{P-claim: } S, c_p, t, H(m), SIG_S(H(H(m)|S|c_p|t)). \qquad (1)$$

Here, t is the current time. c_p ($c_p \in [1, L]$) is the packet count of S, which means that this is the c_p^{th} new packet S has created and sent to the network in the current time interval. S increases c_p by one after sending m out.

The P-claim is attached to packet m as a header field, and will always be forwarded along with the packet to later hops. When the contacted node receives this packet, it verifies the signature in the P-claim, and checks the value of c_p. If c_p is larger than L, it discards this packet; otherwise, it stores this packet and the P-claim.

4.1.2 T-Claim

When node A transmits a packet m to node B, it appends a T-claim to m. The T-claim includes A's current transmission count c_t for m (i.e., the number of times it has transmitted m out) and the current time t. The T-claim is

$$\text{T-claim: } A, B, H(m), c_t, t, SIG_A(H(A|B|H(m)|c_t|t)). \qquad (2)$$

B checks if c_t is in the correct range based on if A is the source of m. If c_t has a valid value, B stores this T-claim.

In single-copy and multicopy routing, after forwarding m for enough times, A deletes its own copy of m and will not forward m again.

4.2 Inconsistency Caused by Attack

In a dishonest P-claim, an attacker uses a smaller packet count than the real value. (We do not consider the case where the attacker uses a larger packet count than the real value, since it makes no sense for the attacker.) However, this packet count must have been used in another P-claim generated earlier. This causes an inconsistency called *count reuse*, which means the use of the same count in two different P-claims generated by the same node. For example in Fig. 3a the count value 3 is reused in the P-claims of packet $m3$ and $m4$. Similarly, count reuse is also caused by dishonest T-claims.

4.3 Protocol

Suppose two nodes contact and they have a number of packets to forward to each other. Then our protocol is sketched in Algorithm 1.

Algorithm 1. The protocol run by each node in a contact

Metadata (P-claim and T-claim) exchange and attack detection
if Have packets to send **then**
  For each new packet, generate a P-claim;
  For all packets, generate their T-claims and sign them with a hash tree;
  Send every packet with the P-claim and T-claim attached;
end if
if Receive a packet **then**
  if Signature verification fails **or** the count value in its P-claim or T-claim is invalid **then**

```
 9:         Discard this packet;
10:     end if
11:     Check the P-claim against those locally collected and
        generated in the same time interval to detect
        inconsistency;
12:     Check the T-claim against those locally collected for
        inconsistency;
13:     if Inconsistency is detected then
14:         Tag the signer of the P-claim (T-claim, respec-
            tively) as an attacker and add it into a blacklist;
15:         Disseminate an alarm against the attacker to the
            network;
16:     else
17:         Store the new P-claim (T-claim, respectively);
18:     end if
19: end if
```

When a node forwards a packet, it attaches a T-claim to the packet. Since many packets may be forwarded in a contact and it is expensive to sign each T-claim separately, an efficient signature construction is proposed in Section 4.7. The node also attaches a P-claim to the packets that are generated by itself and have not been sent to other nodes before (called *new packet* in line 3, Algorithm 1).

When a node receives a packet, it gets the P-claim and T-claim included in the packet. It checks them against the claims that it has already collected to detect if there is any inconsistency (see Section 4.5). Only the P-claims generated in the same time interval (which can be determined by the time tag) are cross-checked. If no inconsistency is detected, this node stores the P-claim and T-claim locally (see Section 4.4).

To better detect flood attacks, the two nodes also exchange a small number of the recently collected P-claims and T-claims and check them for inconsistency. This metadata exchange process is separately presented in Section 5.

When a node detects an attacker, it adds the attacker into a blacklist and will not accept packets originated from or forwarded by the attacker. The node also disseminates an alarm against the attacker to other nodes (see Section 4.6).

4.4 Local Data Structures

Each node collects P-claims and T-claims from the packets that it has received and stores them locally to detect flood attacks. Let us look at a received packet m and the P-claim and T-claim included in this packet. Initially, this pair of P-claim and T-claim are stored in full with all the components shown in Formulas 1 and 2. When this node removes m from its buffer (e.g., after m is delivered to the destination or dropped due to expiration), it compacts this pair of claims to reduce the storage cost. If this pair of claims have been sampled for metadata exchange, they will be stored in full until the exchange process ends and be compacted afterward.

4.4.1 Compact P-Claim Storage

Suppose node W stores a P-claim $\mathbb{C}_P = \{S, c_p, t, H(m), SIG_S\}$. It compacts the P-claim as follows: using the timestamp t, W gets the index i of the time interval that t belongs to. The signature is discarded since it has been verified. The compacted P-claim is

$$S, i, c_p, \widetilde{H}_8, \tag{3}$$

where $\widetilde{H}_8$ is a 8-bit string called *hash remainder*. It is obtained by concatenating 8 random bits of the packet hash $H(m)$. The indices of these 8 bits in the hash are determined by eight locators. The locators are randomly and independently generated by W for S at the beginning of the ith interval, and are shared by all the P-claims issued by S in the ith interval. Each locator only has $\log_2 h$ bits where h denotes the size of a hash (e.g., 256 for SHA-256). W keeps these locators secret.

Suppose node W has collected n P-claims generated by S in interval i. For all these claims, only one source node ID and interval index is stored. Also, instead of directly storing the packet count values c_p included in these P-claims, the compact structure uses a L-bit long bit vector to store them. If value c appears in these P-claims, the cth bit of the vector is set. Let $\mathcal{C}_S^i$ denote the compact structure of these P-claims. Then

$$\mathcal{C}_S^i = S, i, \textit{bit-vector}, \textit{locators}, [\widetilde{H}_{8_1}, \widetilde{H}_{8_2}, \ldots, \widetilde{H}_{8_n}]. \tag{4}$$

4.4.2 Compact T-Claim Storage

Suppose node W stores a T-claim $\mathbb{C}_T = \{R, W, H(m), c_t, t, SIG_R\}$ issued by node R. The signature is discarded since it has been verified. W does not need to store its own ID and t is not useful for inconsistency check. Then the compacted T-claim is

$$R, c_t, \widetilde{H}_{32}, \tag{5}$$

where $\widetilde{H}_{32}$ is a 32-bit hash remainder defined similarly as $\widetilde{H}_8$. Suppose W has collected n T-claims generated by R. Then the compact structure of these T-claims is

$$\mathcal{C}_R = R, \textit{locators}, [\widetilde{H}_{32_1}, c_{t_1}], \ldots, [\widetilde{H}_{32_n}, c_{t_n}]. \tag{6}$$

The locators are randomly and independently generated by W for R, and are shared by all the T-claims issued by R.

4.5 Inconsistency Check

Suppose node W wants to check a pair of P-claim and T-claim against its local collections to detect if there is any inconsistency. The inconsistency check against full claims is trivial: W simply compares the pair of claims with those collected. In the following, we describe the inconsistency check against compactly stored claims.

4.5.1 Inconsistency Check with P-Claim

From the P-claim node W gets: the source node ID S, packet count c_p, timestamp t, and packet hash H. To check inconsistency, W first uses S and t to map the P-claim to the structure $\mathcal{C}_S^i$ (see (4)). Then it reconstructs the hash remainder of H using the locators in $\mathcal{C}_S^i$. If the bit indexed by the packet count c_p is set in the bit-vector but the hash remainder is not included in $\mathcal{C}_S^i$, count reuse is detected and S is an attacker.

The inconsistency check based on compact P-claims does not cause false positive, since a good node never reuses any count value in different packets generated in the same interval. The inconsistency check may cause false negative if the two inconsistent P-claims have the same hash remainder. However, since the attacker does not know which bits constitute the hash remainder, the probability of false negative is only 2^{-8}. Thus, it has minimal effect on the overall detection probability.

4.5.2 Inconsistency Check with T-Claim

From the T-claim node W gets: the sender ID R, receiver ID Q and transmission count c_t. If Q is W itself (which is possible if the T-claim has been sent out by W but returned by an attacker), W takes no action. Otherwise, it uses R to map the T-claim to the structure $\mathcal{C}_R$ (see (6)). If there is a 2-tuple $[\widetilde{H}'_{32}, c'_t]$ in $\mathcal{C}_R$ that satisfies 1) $\widetilde{H}'_{32}$ is the same as the remainder of H, and 2) $c'_t = c_t$, then the issuer of the T-claim (i.e., R) is an attacker.

The inconsistency check based on compact T-claims does not cause extra false negative. False positive is possible but it can be kept low as follows: node W may falsely detect a good node R as an attacker if it has received two T-claims generated by R that satisfy two conditions: 1) they are generated for two different packets, and 2) they have the same hash remainder. For 32-bit hash remainder, the probability that each pair of T-claims lead to false detection is 2^{-32}. In most cases, we expect that the number of T-claims generated by R and received by W is not large due to the opportunistic contacts of DTNs, and thus the probability of false detection is low. As W receives more T-claims generated by R, it can use a longer (e.g., 64-bit) hash remainder for R to keep the probability of false detection low. Moreover, such false detection is limited to W only, since W cannot convince other nodes to accept the detection with compact T-claim.

4.6 Alarm

Suppose in a contact a node receives a claim $\mathbb{C}_r$ from a forwarded data packet or from the metadata exchange process (see Section 5.3) and it detects inconsistency between $\mathbb{C}_r$ and a local claim $\mathbb{C}_l$ that the node has collected. $\mathbb{C}_r$ is a full claim as shown in Formula 1 (or 2), but $\mathbb{C}_l$ may be stored as a full claim or just a compact structure shown in Formula 3 (or 5).

If $\mathbb{C}_l$ is a full claim, the node can broadcast (via Epidemic routing [28]) a *global alarm* to all the other nodes to speed up the attacker detection process. The alarm includes the two full claims $\mathbb{C}_l$ and $\mathbb{C}_r$. When a node receives an alarm, it verifies the inconsistency between the two included claims and their signatures. If the verification succeeds, it adds the attacker into its blacklist and broadcasts the alarm further; otherwise, it discards the alarm. The node also discards the alarm if it has broadcast another alarm against the same attacker.

If the detecting node stores $\mathbb{C}_l$ as a compact structure, it cannot convince other nodes to trust the detection since the compact structure does not have the attacker's signature. Thus it cannot broadcast a global alarm. However, since the attacker may have reused the count value of $\mathbb{C}_r$ to other claims besides $\mathbb{C}_l$, the detecting node can disseminate a *local alarm* that only contains $\mathbb{C}_r$ to its contacted nodes who have received those claims. These contacted nodes can verify the inconsistency between $\mathbb{C}_r$ and their collected claims, and also detect the attacker. If any of these nodes still stores a full claim inconsistent with $\mathbb{C}_r$, it can broadcast a global alarm as done in the previous case; otherwise, it disseminates a local alarm. As this iterative process proceeds, the attacker can be quickly detected by many nodes. Each node only disseminates one local alarm for each detected attacker.

A local alarm and a global alarm against the same attacker may be disseminated in parallel. If a node receives the global alarm first and then receives the local alarm, it discards the local alarm. If it receives the local alarm first, when it receives the global alarm later, it discards the local alarm and keeps the global alarm.

An attacker may falsify an alarm against a good node. However, since it does not have the node's private key (as our assumption), it cannot forge the node's signatures for the claims included in the alarm. Thus, the alarm will be discarded by other nodes and this attack fails.

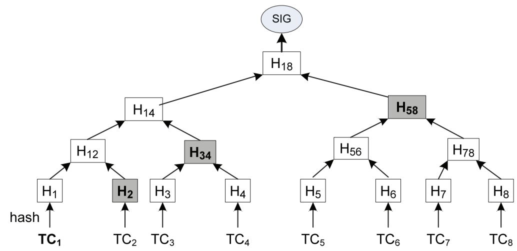


Fig. 5. The Merkle hash tree constructed upon eight T-claims $TC_1, \ldots, TC_8$. In the tree, H_i is the hash of TC_i, and an inner node is the hash of its child nodes. The signature of TC_1 includes H_2, H_{34}, H_{58}, and SIG.

4.7 Efficient T-Claim Authentication

The T-claims of all the packets transmitted in a contact should be signed by the transmitting node. Since the contact may end at any unpredictable time, each received T-claim must be individually authenticated. A *naive* approach is to protect each T-claim with a separate public-key signature, but it has high computation cost in signature generation and verification.

Our scheme uses Merkle hash tree [29] to amortize the computation cost of public-key-based signature on all the T-claims that the node sends out in a contact. Specifically, after a node generates the T-claims (without signature) for all the packets it want to send, it constructs a hash tree upon these partial T-claims, and signs the root of the tree with a public-key-based signature. Then the signature of a T-claim includes this root signature and a few elements of the tree. Fig. 5 shows the hash tree constructed upon eight T-claims and the tree elements included in the signature of the first T-claim. We refer to the original paper [29] for details. In this way, for all the T-claims sent by the sender in a contact, only one public-key based signature is generated by the sender and verified by the receiver.

4.8 Dealing with Different Rate Limits

Previously we have assumed that all nodes have the same rate limit L. When nodes have different rate limits, for our detection scheme to work properly, each intermediate node that receives a packet needs to know the rate limit L of the source of the packet, such that it can check if the packet count is in the correct range $1, 2, \ldots, L$. To do so, when a source node sends out a packet, it attaches its rate limit certificate to the packet. The intermediate nodes receiving this packet can learn the node's authorized rate limit from the attached certificate.

4.9 Replica Flood Attacks in Quota-Based Routing Protocols

Our scheme to detect replica flood attacks can also be adapted to quota-based routing protocols [23], [19], [24].

Quota-based routing works as follows: each node has a quota for each packet that it buffers, and the quota specifies the number of replicas into which the current packet is allowed to be split. When a source node creates a packet, its quota for the packet is L' replicas, where L' is a system parameter. When the source contacts a relay node, it can split multiple replicas to the relay according to the quality of the relay. After the split, the relay's quota for the packet is the number of replicas split to it, and the source node's quota is reduced by the same amount. This procedure continues recursively, and each node carrying the packet can split out a number of replicas less than its current quota for the packet. It can be seen that each packet can simultaneously have at most L' replicas in the network.

In quota-based routing, replica flood attacks (where an attacker sends out more replicas of a packet than its quota) can be detected by our approach as follows.

First, we observe that quota-based routing (with the total quota determined at the source) can be emulated by single-copy routing if different replicas of the same packet appear different to intermediate nodes and each replica is forwarded in a similar way as single-copy routing. A node can split multiple replicas of a packet to another node, but it can only send each replica out once. For instance, if a node has forwarded Replica 1 to one relay, it must remove Replica 1 from its local buffer, and it cannot forward this replica again to another relay.

To differentiate replicas, the source assigns a unique index to each replica as a header field, and signs the replica to prevent intermediate nodes from modifying the index. The index value should be in range $[1, L']$, and replicas with invalid index will be discarded. In this way, a node's local quota for a packet is represented by the number of replicas (with different indices) that it buffers. Note that an intermediate node cannot increase its quota by forging replicas since it does not have the source node's key to generate a valid signature.

To prevent a node from abusing its quota, we need to ensure that the node only forwards each replica once. T-claim can be used to achieve this goal. Particularly, when a node splits multiple replicas of a packet to another node, it generates a T-claim for each replica. The inconsistency check (see Section 4.5) can be applied here to detect the attackers that transmit the same replica more than once.

5 Metadata Exchange

When two nodes contact they exchange their collected P-claims and T-claims to detect flood attacks. If all claims are exchanged, the communication cost will be too high. Thus, our scheme uses sampling techniques to keep the communication cost low. To increase the probability of attack detection, one node also stores a small portion of claims exchanged from its contacted node, and exchanges them to its own future contacts. This is called redirection.

5.1 Sampling

Since P-claims and T-claims are sampled together (i.e., when a P-claim is sampled the T-claim of the same packet is also sampled), in the following we only consider P-claims.

A node may receive a number of packets (each with a P-claim) in a contact. It randomly samples Z (a system parameter) of the received P-claims, and exchanges the sampled P-claims to the next K (a system parameter) different nodes it will contact, excluding the sources of the P-claims and the previous hop from which these P-claims are received.

However, a vulnerability to tailgating attack should be addressed. In tailgating attack, one or more attackers tailgate a good node to create a large number (say, d) of frequent contacts with this node, and send Z packets (not necessarily generated by the attackers) to this node in each created contact. If this good node sends the Zd P-claims of these contacts to the next K good nodes it contacts, much effective bandwidth between these good nodes will be wasted, especially in a large network where K is not small.

To address this attack, the node uses an inter-contact sampling technique to determine which P-claims sampled in historical contacts should be exchanged in the current contact. Let $\mathcal{S}_K$ denote a set of contacts. This set includes the minimum number of most recent contacts between this node and at least K other different nodes. Within this set, all the contacts with the same node are taken as one single contact and a total of Z P-claims are sampled out of these contacts. This technique is not vulnerable to the tailgating attack since the number of claims exchanged in each contact is bounded by a constant.

5.2 Redirection

There is a stealthy attack to flood attack detection. For replica flood attacks, the condition of detection is that at least two nodes carrying inconsistent T-claims can contact. However, suppose the attacker knows that two nodes A and B never contact. Then, it can send some packets to A, and invalidly replicate these packets to B. In this scenario, this attacker cannot be detected since A and B never contact. Similarly, the stealthy attack is also harmful for some routing protocols like Spray-and-Wait [19] in which each packet is forwarded from the source to a relay and then directly delivered from the relay to the destination.

To address the stealthy attack, our idea is to add one level of indirection. A node redirects the Z P-claims and T-claims sampled in the current contact to one of the next K nodes it will contact, and this contacted node will exchange (but not redirect again) these redirected claims in its own subsequent contacts. Look at the example in Fig. 6. Suppose attacker S sends mutually inconsistent packets to two nodes A and B which will never contact. Suppose A and B redirect their sampled P-claims to node C and D, respectively. Then so long as C and B or D and A or C and D can contact, the attack has a chance to be detected. Thus, the successful chance of stealthy attack is significantly reduced.

5.3 The Exchange Process

Each node maintains two separate sets of P-claims (T-claims, respectively in the following) for metadata exchange, a

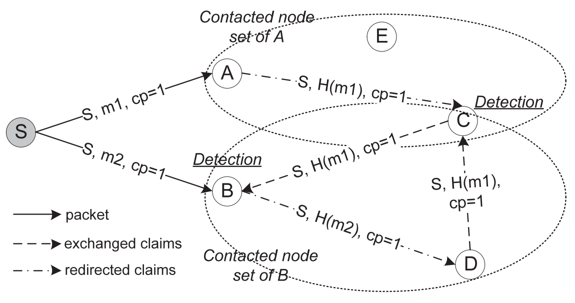


Fig. 6. The idea of redirection which is used to mitigate the stealthy attack.

sampled set which includes the P-claims sampled from the most recent contacts with K different nodes (i.e., $\mathcal{S}_K$ in Section 5.1), and a *redirected set* which includes the P-claims redirected from those contacts. Both sets include Z P-claims obtained in each of those contacts.

When two nodes A and B contact, they first select KZ P-claims from each set with the inter-contact sampling technique (see Section 5.1), and then send these P-claims to each other. When A receives a P-claim, it checks if this P-claim is inconsistent with any of its collected P-claims using the method described in Section 4.5. If the received P-claim is inconsistent with a locally collected one and the signature of the received P-claim is valid, A detects that the issuer (or signer) of the received P-claim is an attacker.

Out of all the P-claims received from B, A randomly selects Z of the P-claims from the sampled set of B, and stores them to A's redirected set. All other P-claims received from B are discarded after inconsistency check.

5.4 Metadata Deletion

A node stores the P-claims and T-claims collected from received data packets for a certain time denoted by τ and deletes them afterward. It deletes the claims redirected from other nodes immediately after it has exchanged them to K different nodes.

6 Analysis

This section presents rigorous analysis over the security and cost of our scheme, and discusses the optimal parameter to maximize the effectiveness of flood attack detection under a certain amount of exchanged metadata per contact.

6.1 Detection Probability

The following analysis assumes uniform and independent contacts between nodes, i.e., at any time each node's next contacted node can be any other node with the same probability. This assumption holds for mobility models such as Random Waypoint (RWP) where the contacts between all node pairs can be modeled as i.i.d. Poisson processes [30]. When analyzing the detection probability, we assume that each attacker acts alone. The case of collusion is analyzed separately in Section 6.4.

6.1.1 The Basic Attack

First we consider a basic attack (see Fig. 7a) in which an attacker S floods two sets of mutually inconsistent packets to two good nodes A and B, respectively. Each flooded packet received by A is inconsistent with one of the flooded packets received by B. In the contacts with A and B, S also forwards some normal, not flooded, packets to A and B to make the attack harder to detect. Let y denote the proportion of flooded packets among those sent by S. For simplicity, we assume y is the same in both contacts. Suppose A and B redirect the claims sampled in the contact with S to C and D, respectively.

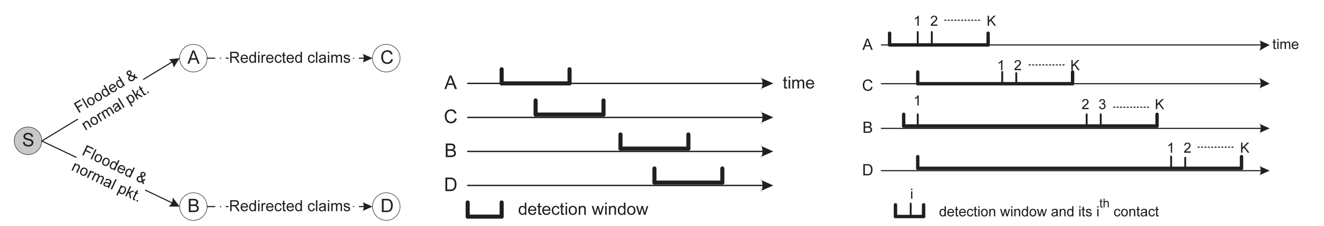
To consider the worst case performance, suppose the flooded packets are not forwarded from A and B to other nodes (which is the case in Spray-and-Wait [19]), i.e., only A and B have the inconsistent claims. Note that the analysis also applies to the detection of replica flood attacks.

For convenience, we define node A's (or B's) *detection window* as from the time it receives the flooded packets to the time it exchanges the sampled claims to K nodes, and node C's (or D's) detection window as from the time it receives the redirected claims to the time it exchanges them to K nodes. The attacker has a chance to be detected if node pairs $\langle A, B\rangle$, $\langle A, D\rangle$, $\langle C, B\rangle$ and $\langle C, D\rangle$ can contact within their detection windows. Table 1 shows the variables used in the analysis.

Lower bound. The lower bound of detection probability is obtained in the following scenario (see Fig. 7b): when B receives the packets from S, both A and C have finished their detection window. Due to the effect of sampling, the attacker can be detected

1. by A if $A \in \mathcal{S}_B$ and $e_B = TRUE$;
2. by A if D is a good node, $A \in \mathcal{S}_D$ and $e_B = TRUE$;
3. by C if C is a good node, $C \in \mathcal{S}_B$ and $\hat{e}_{AB} = TRUE$; or
4. by C if both C and D are good nodes, $C \in \mathcal{S}_D$ and $\hat{e}_{AB} = TRUE$.

Since each of A and B exchanges the sampled claims to K nodes other than itself, and C (D) exchanges the



(a) Considered basic attack (b) Lower bound scenario (c) Upper bound scenario.

Fig. 7. (a) The basic attack considered for detection probability analysis. Attacker S floods packets to A and then to B. (b) The scenario when the lower bound detection probability can be obtained : When B receives the flooded packets from S, both A and C have finished their detection window. (c) The scenario when the upper bound detection probability can be obtained: D receives the redirected claims from B not later than the time when C receives the redirected claims from A, and they are the first node that A and B encounter after the contact with S.

TABLE 1
Variables Used in the Analysis

$\mathcal{S}_i$	The K nodes that node i ($i \in \{A, B, C, D\}$) exchanges its sampled or redirected claims to.
e_i	A boolean event which means if node i ($i \in \{A, B\}$) has sampled at least one flooded packet ($e_i = TRUE$) or not ($e_i = FALSE$)
$\hat{e}_{AB}$	A boolean event which means if node A and B have sampled at least one pair of inconsistent packets ($\hat{e}_{AB} = TRUE$) or not
P_s	The probability that event $e_A = TRUE$ (or $e_B = TRUE$, resp.)
P_{ovp}	The conditional probability that if $e_A = e_B = TRUE$ then $\hat{e}_{AB} = TRUE$.
P_d	The probability that S is detected.
P_d^l	The lower-bound of P_d.
P_d^u	The upper-bound of P_d.
N	The number of nodes in the network.
M	The number of attackers in the network.
r	The proportion of good nodes, i.e., $r = \frac{N-M}{N}$.
K	The number of nodes that a claim is exchanged to. $K \ll N$.
a	The number of flooded packets sent to A and B.
n	The total number of packets sent to A and B.
y	The proportion of flooded packets sent to A and B, i.e., $y = a/n$.

redirected claims to K nodes other than A (B), according to the uniform contact assumption, we have

$$P_d = P_s\Bigg[1 - \left(1 - \frac{K}{N-1}\right) \cdot \left(1 - r\frac{K}{N-2}\right) \cdot \left(1 - r\frac{K}{N-1}P_sP_{ovp}\right) \cdot \left(1 - r^2\frac{K}{N-2}P_sP_{ovp}\right)\Bigg]. \tag{7}$$

The expected number of flooded packets that A or B can sample is yZ. Since Z is typically small while a is not that small (which we believe realistic), P_{ovp} is negligible. Considering that $K \ll N$, P_d is approximated as follows:

$$P_d \approx P_s\left[1 - \left(1 - \frac{K}{N-1}\right)\left(1 - r\frac{K}{N-2}\right)\right] \approx P_s\frac{1+r}{N}K.$$

Since random sampling is used, it is trivial to get

$$P_s = 1 - \frac{n-a}{n}\frac{n-a-1}{n-1}\cdots\frac{n-a-Z+1}{n-Z+1} \geq 1 - (1-a/n)^Z = 1 - (1-y)^Z.$$

Thus, we have $P_d \geq K\frac{1+r}{N}(1-(1-y)^Z)$, and a lower bound of the detection probability is

$$P_d^l = K\frac{1+r}{N}(1-(1-y)^Z). \tag{8}$$

Upper bound. The upper bound of detection probability is obtained in the following scenario (see Fig. 7c): D receives the redirected claims from B not later than the time C receives the redirected claims from A, and they are the first node that A and B encounter after the contact with S. Besides the four cases that we discussed when deriving the lower bound, the attacker can also be detected

1. by B if $B \in \mathcal{S}_A$ and $e_A = TRUE$;
2. by B if C is a good node, $B \in \mathcal{S}_C$ and $e_A = TRUE$;
3. by D if D is a good node, $D \in \mathcal{S}_A$ and $\hat{e}_{AB} = TRUE$; or
4. by D if both C and D are good nodes, $D \in \mathcal{S}_C$ and $\hat{e}_{AB} = TRUE$.

Similarly as in the lower bound case, we can obtain that the detection probability is $P_d \approx 2K\frac{1+r}{N}P_s$. Since $P_s \leq 1$, an upper bound of the detection probability is

$$P_d^u = \frac{2K(1+r)}{N}. \tag{9}$$

6.1.2 Stronger Attacks

We use the number of times that a count value is reused to represent the strength of an attack. Intuitively, if each count value is used many times, the attacker floods a lot of packets or replicas. Let X denote the number of times a count value is reused. We want to derive the relation between X and the probability that the attacker will be detected.

The stronger attacks we consider extends the basic attack (see Section 6.1.1) as follows: suppose the attacker has sent a set of packets to a good node G_0, and then it successively sends the same number of packets (reusing the count values of the packets sent to G_0) to X good nodes $G_1, \ldots, G_X$. All other parameters in the basic analysis apply here. From a global point of view, a total of $\frac{X(X+1)}{2}$ node pairs have inconsistent packets, and each pair can detect the attacker independently as analyzed in Section 6.1.1. Then the attacker will be detected by at least one node pair with a probability

$$P_d^X = 1 - (1 - P_d)^{\frac{X(X+1)}{2}}. \tag{10}$$

We can see that the stronger the attack is, the more likely the attacker will be detected.

6.2 Cost Analysis

6.2.1 Communication

The communication cost mainly has two parts. One part is the P-claim and T-claim transmitted with each packet, and the other part is the partial claims transmitted during metadata exchange. As to the latter, at most $4ZK$ P-claims and $4ZK$ T-claims are exchanged in each contact, with one half for sampled and the other half for redirected claims.

6.2.2 Computation

As to signature generation, a node generates one signature for each newly generated packet. It also generates one signature for all its T-claims as a whole sent in a contact. As to signature verification, a node verifies the signature of each received packet. It also verifies one signature for all the T-claims as a whole received in one contact.

6.2.3 Storage

Most P-claims and T-claims are compacted when the packets are forwarded. The Z sampled P-claims and T-claims are stored in full until the packets are forwarded or have been exchanged to K nodes, whichever is later, and then compacted. For each received packet, less than 20 bytes of compact claims are stored for time duration τ.

6.3 Parameter Selection with Fixed Cost of Metadata Exchange

We study the following question: if we fix the communication cost of metadata exchange (note that little can be done with the transmission of claims within packets), then how should we set parameter K and Z to maximize the detection probability? Note that the communication cost of metadata exchange is proportional to ZK. As to detection probability, we consider the lower-bound detection probability for the basic attack which can be written as $P_d = cK(1-(1-y)^Z)$.

Lemma 1. *If the communication cost of metadata exchange is fixed at $ZK = C$, then P_d is maximized at $K = C$ and $Z = 1$.*

Proof. P_d can be rewritten as $P_d = cC\frac{1-(1-y)^Z}{Z}$. The derivative of P_d over Z is

$$P_d'(Z) = \frac{cC}{Z^2}[(1-y)^Z(1-\ln(1-y)^Z)-1]. \qquad (11)$$

Let $u = (1-y)^Z$ $(0 < u \leq 1)$, then $P_d'(Z) = \frac{cC}{Z^2}(u - u\ln u - 1)$. Let $g(u) = u - u\ln u - 1$. Then $g'(u) = -\ln u \geq 0$, which means $g(u)$ monotonically increases. Since $g(1) = 0, g(u) \leq 0$ when $0 < u \leq 1$. Therefore, $P_d'(Z) \leq 0$, which means P_d monotonically decreases with Z. Thus, to maximize P_d, Z should be set the minimum value 1. □

Remarks. In this parameter setting, the lower bound detection probability can be written as $P_d = yK\frac{1+r}{N}$. Suppose the attacker launches attacks independently. Then it can be detected after $\frac{N}{yK(1+r)}$ attacks. If the attacker wants to stay undetected for a longer time, it should maintain a smaller y, which means the attack effect is weaker; if it wants to make a big attack impact, it should maintain a high y, but this means it will be detected in a shorter time. From another point of view, since the attacker only uses y proportion its capacity for flood attack, it is equivalent that the attacker can attack at full capacity for only $\frac{N}{K(1+r)}$ contacts. Thus, the attacks can be effectively mitigated.

6.4 Collusion Analysis

6.4.1 Packet Flood Attack

One attacker may send a packet with a dishonest packet count to its colluder, which will forward the packet to the network. Certainly, the colluder will not exchange the dishonest P-claim with its contacted nodes. However, so long as the colluder forwards this packet to a good node, this good node has a chance to detect the dishonest claim as well as the attacker. Thus, the detection probability is not affected by this type of collusion.

6.4.2 Replica Flood Attack

When attackers collude, they can inject invalid replicas of a packet without being detected, but the number of flooded replicas is effectively limited in our scheme. More specifically, in our scheme for a unique packet all the M colluders as a whole can flood a total of $M-1$ invalid replicas without being detected. To the contrast, when there is no defense, a total of $N-M$ invalid replicas can be injected by the colluders for each unique packet. Since the number of colluders is not very large, our scheme can still effectively mitigate the replica flood attack. This will be further evaluated in Section 7.

7 Performance Evaluations

7.1 Experiment Setup

To evaluate the performance and cost of our scheme, we run simulations on a synthetic trace generated by the Random Waypoint [30] mobility model and on the MIT Reality trace [17] collected from the real world.

In the synthetic trace, 97 nodes move in a 500×500 square area with the RWP model. The moving speed is randomly selected from $[1, 1.6]$ to simulate the speed of walking, and the transmission range of each node is 10 to simulate that of Bluetooth. Each simulation lasts 5×10^5 time units.

The MIT Reality trace [17] has been shown [31], [32] to have social community structures. Ninety seven Smart phones are carried by students and staff at MIT over 10 months. These phones run Bluetooth device discovery every 5 minutes and log about 110 thousand contacts. Each logged contact includes the two contact parties, the start time and duration of the contact.

In the simulations, 20 percent of nodes are deployed as attackers. They are randomly deployed or selectively deployed to high-connectivity nodes. The buffer size of each node is 5 MB, the Drop Tail policy is used when buffer overflows. The bandwidth is 2 Mbps. Each node generates packets of 10 KB with random destinations at a uniform rate. Parameter $Z = 1$.

7.2 Routing Algorithms and Metrics

We use the following routing protocols in evaluations:

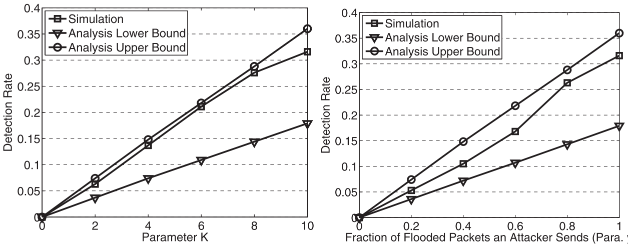
- **Forward.** A single-copy routing protocol where a packet is forwarded to a relay if the relay has more frequent contacts with the destination.
- **SimBet [8].** A single-copy routing protocol where a packet is forwarded to a relay if the relay has a higher simbet metric, which is calculated from two social measures (similarity and betweenness).
- **Spray-and-wait [19].** A multicopy protocol, where the source replicates a packet to $L' = 3$ relays and each relay directly delivers its copy to the destination when they contact.
- **Spray-and-focus [19].** It is similar to Spray-and-Wait, but each packet copy is individually routed to the destination with Forward.
- **Propagation.** A packet is replicated to a relay if the relay has more frequent contacts with the destination.

We use the following performance evaluation metrics:

- **Detection rate.** The proportion of attackers that are detected out of all the attackers.
- **Detection delay.** From the time the first invalid packet is sent to the time the attacker is detected.
- **Computation cost.** The average number of signature generations and verifications per contact.
- **Communication cost.** The number of P-claim/T-claim pairs transmitted into the air, normalized by the number of packets transmitted.
- **Storage cost.** The time-averaged kilobytes stored for P-claims and T-claims per node.

7.3 Analysis Verification

We use the synthetic trace to verify our analysis results given in Section 6, since in this trace the contacts between node pairs are i.i.d. [30] which conforms to our assumption for the analysis. We divide the trace into 10 segments, each with 5×10^4 time units, and run simulations on each of the third-seventh segments three times with different random seeds. Each data point is averaged over the individual runs. Spray-and-Wait is used as the routing protocol to consider the worst case of packet flood detection (see Section 6.1.1).



(a) Effect of Parameter K (b) Strength of Attack (Para. y)

Fig. 8. Verification of analysis results on the synthetic trace. Spray-and-Wait is used as the routing protocol. Each attacker launches the basic attack once.

Here we only verify the detection probability for the basic attack, since the detection probability for the strong attack can be derived from it in a straightforward way. In this group of simulations, each attacker launches the basic attack once. It sends out two sets of packets to two good nodes with 10 packets in each set (i.e., $n = 10$), and these two sets contain mutually inconsistent packets. We first fix parameter $y = 1.0$ (see Table 1) but change parameter K from 0 to 10, and then we fix parameter $K = 10$ but change y from 0 to 1.0. The results are shown in Figs. 8a and 8b, respectively. It can be seen that the simulation results are between the analytical lower bound and upper bound, which verifies the correctness of our analysis.

7.4 Detection Rate

The Reality trace is used. We divide the trace into segments of one month, and run simulations on each of the third-seventh segments three times with different random seeds. Each data point is averaged over the individual runs. By default, each attacker launches the basic attack once, and it floods one packet out (i.e., $n = 1$, $y = 1.0$). By default, attackers are selectively deployed to high-connectivity nodes.

Fig. 9a shows the effect of parameter K in different routing protocols. Generally speaking, when K increases, the detection rate also increases because the inconsistent packets are exchanged to more nodes and have more chances to be detected. When $K = 0$, no attacker is detected in Spray-and-Wait, since no metadata is exchanged for detection. However, attackers can still be detected in the other three algorithms, because the inconsistent packets are forwarded to multiple nodes and the node that receives two inconsistent packets can detect the attacker. Among these protocols, Propagation achieves the highest detection rate since it replicates inconsistent packets the most number of times. Between the two single-copy routing protocols, SimBet has a higher detection rate than Forward. This is because SimBet tends to forward packets to the more socially connected nodes and thus these nodes are more likely to collect inconsistent packets.

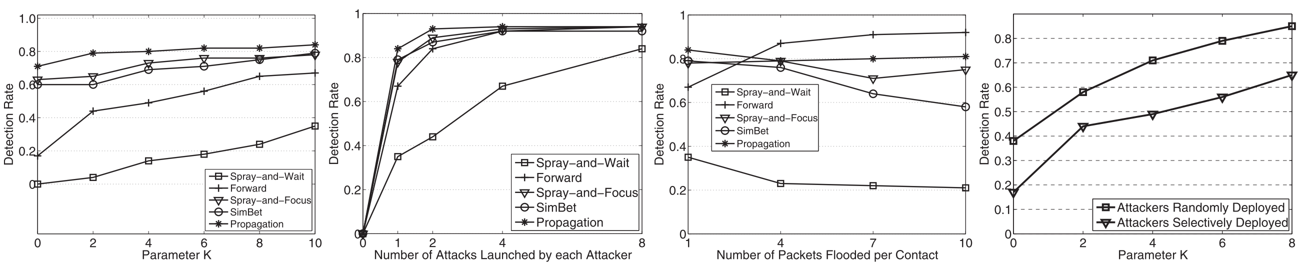
Fig. 9b shows the results when each attacker launches the basic attack independently for a varying number of times. As the attackers launch more attacks, the detection rate quickly increases for obvious reasons.

Fig. 9c shows the effect of the number of packets that an attacker floods in each contact (i.e., parameter n). As an attacker floods more packets in each contact, the detection rate decreases in Spray-and-Wait and SimBet, increases in Forward and does not change much in Spray-and-focus and Propagation. The opposite trends are due to two factors that affect the detection rate reversely. On the one hand, sampling decreases detection rate. To explain this more clearly, let us look at the basic attack scenario in Fig. 7a for Spray-and-Wait. Since $Z = 1$, A (B, respectively) only samples one packet out of all the packets received from the attacker and redirects it to C (D, respectively). When $n = 1$, C and D will receive mutually inconsistent claims, which means in (7) $P_{ovp} = 1.0$. However, when n is larger than 1, C and D may not receive a pair of inconsistent claims due to the independent sampling by A and B. As n increases, P_{ovp} decreases and thus the detection rate also decreases. On the other hand, for the routing protocols where each packet is forwarded in multiple hops, when an attacker sends more attack packets in each contact, it is more likely that one pair of inconsistent packets are forwarded to the same intermediate node and lead to detection.

Fig. 9d shows the effect of attacker deployment. The detection rate is lower when attackers are selectively deployed to high-connectivity nodes. This is because when attackers are selectively deployed they have more contacts with good nodes. The probability that a good node exchanges its sampled claims to attackers rather than to other good nodes is higher, but attackers do not run detection against each other.

7.5 Detection Delay

Fig. 10 shows the CDF of detection delay when Propagation is used as the routing protocol on the Reality trace. For comparison, the CDF of routing delay (i.e., from the time a packet is generated to the time it is delivered) is also plotted. Here, no lifetime is set for packets. It can be seen that 90 percent of the attacks can be detected by our scheme



(a) Parameter K (b) Number of Attacks (c) Packets Flooded per Contact (d) Attacker Deployment

Fig. 9. The detection rate under different conditions. In (d), Forward is used as the routing protocol.

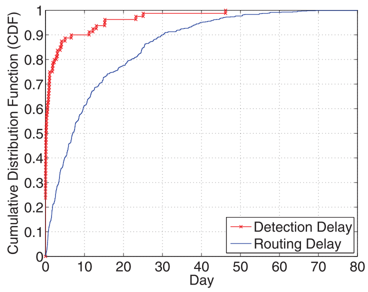


Fig. 10. The detection delay compared with the routing delay of Propagation.

within 10 days. On the contrary, within 10 days only 60 percent of data packets can be delivered by the routing protocol. Hence, the detection delay of our scheme is much lower than the routing delay.

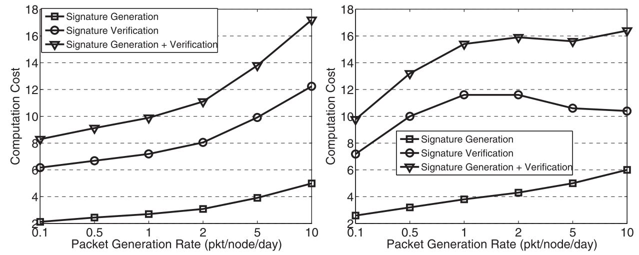
7.6 Undetected Flooded Replicas under Collusion

As mentioned in Section 6.4.2, colluders can flood a small number of replicas without being detected. To evaluate their effect, we run simulations on the Reality trace when all attackers collude. The simulation settings are the same as in Section 2.2. We compare our scheme with the case of no defense. As shown in Fig. 11, even when 20 percent of nodes are attackers and collude, our scheme can still limit the percentage of wasted transmissions to 14 percent in single-copy routing (SimBet) and 6 percent in multicopy routing (Spray-and-Focus), which is only 1/7-1/5 of the wasted transmissions when there is no defense.

7.7 Cost

To evaluate the cost of our scheme in a steady state (i.e., all attackers have been detected), no attackers are deployed in this group of simulations. The Reality trace is used. Packets are generated between the 61st and 120th day of the trace, and statistics are collected from the 91th day. By default, each node generates two packets per day, parameter τ (i.e., the time a claim is stored) is 30 days and K is 10. In a contact, a node may receive some packets but then immediately drop them due to buffer overflow. In such cases, the transmission of the claims attached to these packets is counted into the communication overhead, and the signature generations for these claims are counted into the computation overhead. Since the receiver does not buffer these packets, it does not store these claims or verify their signatures.

We first evaluate the computation cost of our scheme, and Fig. 12 shows the results. When Forward is used as the routing protocol (see Fig. 12a), as the packet generation rate increases, the computation cost also increases since more packets need to be signed and verified. But the cost is still low, less than 20 signature generations and verifications, when each node generates 10 packets per day. Also, it can be seen that there are less signature generations than verifications. This is because in each contact our scheme only signs P-claims for the newly generated packets (which constitute a very small portion of the packets transmitted), and it generates only one signature in total for the T-claims of all forwarded packets due to the use of authentication tree. When Propagation is used as the routing protocol (see Fig. 12b), similar trends hold. When the packet generation rates crosses 1, the signature verification cost turns to decrease. This is because when the traffic load is high many received packets are dropped due to buffer overflow.



(a) Forward (b) Propagation

Fig. 12. The computation cost of our scheme.

Then we evaluate the communication cost. The communication overhead mainly comes from two sources, the transmission of claims attached to data packets, and the transmission of claims in metadata exchange. The total communication cost and the two components are shown in Fig. 13. When K increases from 2 to 10 (see Fig. 13a), the communication cost caused by meta data exchange increases linearly since each sampled claim is transmitted more times. The communication cost caused by the transmission of claims attached to packets is always 1 due to the normalization. In total, less than two pairs of P-claim/T-claim are transmitted per transmission of data packet. When the packet generation rate increases (see Fig. 13b), the total normalized communication cost decreases, because more data packets are transmitted in each contact but the number of claims transmitted for metadata exchange in each contact does not change with the traffic load. When the packet generation rate is larger than 1, the communication cost is smaller than 3. When the packet

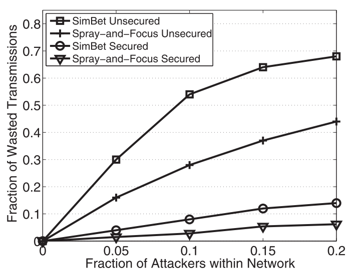
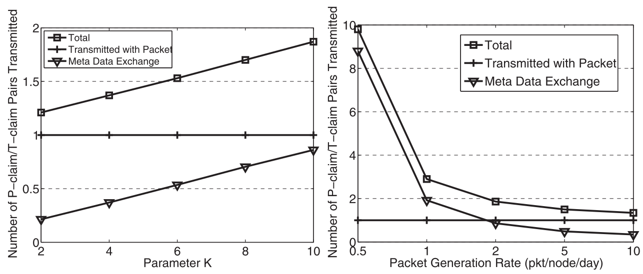


Fig. 11. The effect of undetected replicas on wasted transmissions when attackers collude to launch replica flood attacks.



(a) (b)

Fig. 13. The communication cost of our scheme.

TABLE 2
The Storage (KB) Used for Claims and Data Packets

τ (days)	10	20	30	40	50	-
Claims	67	101	125	139	145	-
Packets	3330	3301	3321	3336	3316	-
Pkt. Generation Rate (pkt/node/day)	0.1	0.5	1	2	5	10
Claims	65	93	113	125	124	114
Packets	334	1572	2596	3321	3716	3808

generation rate is 0.5, the communication cost is higher (i.e., 10). However, at this point the number of packet transmissions is very small, and hence the communication overhead is not an issue. Moreover, since the claims are small in size, they can be attached to and transmitted with data packets and will not incur extra transmissions. Thus, the communication overhead is low.

Finally, we evaluate the storage cost of our scheme against two factors, the time a claim is stored (parameter τ) and the packet generation rate. The results are shown in Table 2. We can see that the storage space used for claims is low, only less than 150 kilobytes per node. This is due to the compact structures we use to store P-claims and T-claims. We noticed that the storage cost does not increase after the packet generation rate reaches two packets per node per day, because when the traffic load is high many received packets are dropped due to buffer overflow.

8 Related Work

Our scheme bears some similarity with previous approaches (e.g., [33]) that detect node clone attacks in sensor networks. Both rely on the identification of some kind of inconsistency to detect the attacker. However, their approaches assumes consistent connectivity between nodes which is unavailable in DTNs. Also, they do not handle the long delays of detection.

A few recent works [10], [25], [12], [11], [13] also address security issues in DTNs. Li et al. [10] studied the blackhole attack in which malicious nodes forge routing metrics to attract packets and drop all received packets. Their approach uses a primitive called encounter ticket to prove the existence of contacts and prevent the forgery of routing metrics, but encounter ticket cannot be used to address flood attacks. Li and Cao [13] also proposed a distributed scheme to mitigate packet drop attacks, which works no matter if the attackers forge routing metrics or not. Ren et al. [11] studied wormhole attacks in DTNs. Chen and Choon [25] proposed a credit-based approach and Shevade et al. proposed a gaming-based approach [12] to provide incentives for packet forwarding. Privacy issues have also be addressed [38], [39],. However, these work do not address flood attacks. Other works (e.g., Sprite [34]) deter abuse by correlating the amount of network resources that a node can use with the node's contributions to the network in terms of forwarding. This approach has been proposed for mobile ad hoc networks, but it is still not clear how the approach can be applied to DTNs, where nodes are disconnected most of the time. Another recent work [14] proposed a batch authentication protocol for DTNs, which verifies multiple packet signatures in an aggregated way to save the computation cost. This work is complementary to ours, and their protocol can also be used in our scheme to further reduce the computation cost of authentication.

Parallel to our work, Natarajan et al. [35] also proposed a scheme to detect resource misuse in DTNs. In their scheme, the gateway of a DTN monitors the activities of nodes and detects an attack if there is deviation from expected behavior. Different from their work that requires a special gateway for counting, our scheme works in a totally distributed manner and requires no special nodes.

9 Conclusions

In this paper, we employed rate limiting to mitigate flood attacks in DTNs, and proposed a scheme which exploits *claim-carry-and-check* to probabilistically detect the violation of rate limit in DTN environments. Our scheme uses efficient constructions to keep the computation, communication and storage cost low. Also, we analyzed the lower bound and upper bound of detection probability. Extensive trace-driven simulations showed that our scheme is effective to detect flood attacks and it achieves such effectiveness in an efficient way. Our scheme works in a distributed manner, not relying on any online central authority or infrastructure, which well fits the environment of DTNs. Besides, it can tolerate a small number of attackers to collude.

Acknowledgments

This work was supported in part by Army Research Office under MURI grant W911NF-07-1-0318.

References

[1] K. Fall, "A Delay-Tolerant Network Architecture for Challenged Internets," *Proc. ACM SIGCOMM*, pp. 27-34, 2003.
[2] P. Hui, A. Chaintreau, J. Scott, R. Gass, J. Crowcroft, and C. Diot, "Pocket Switched Networks and Human Mobility in Conference Environments," *Proc. ACM SIGCOMM*, 2005.
[3] M. Motani, V. Srinivasan, and P. Nuggehalli, "PeopleNet: Engineering a Wireless Virtual Social Network," *Proc. MobiCom*, pp. 243-257, 2005.
[4] J. Burgess, B. Gallagher, D. Jensen, and B. Levine, "Maxprop: Routing for Vehicle-Based Disruption-Tolerant Networks," *Proc. IEEE INFOCOM*, 2006.
[5] S.J.T.U.Grid Computing Center, "Shanghai Taxi Trace Data," http://wirelesslab.sjtu.edu.cn/, 2012.
[6] J. Mirkovic, S. Dietrich, D. Dittrich, and P. Reiher, *Internet Denial of Service: Attack and Defense Mechanisms.* Prentice Hall, 2005.
[7] C. Karlof and D. Wagner, "Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures," *Proc. IEEE First Int'l Workshop Sensor Network Protocols and Applications*, 2003.
[8] E. Daly and M. Haahr, "Social Network Analysis for Routing in Disconnected Delay-Tolerant MANETs," *Proc. MobiHoc*, pp. 32-40, 2007.
[9] W. Gao, Q. Li, B. Zhao, and G. Cao, "Multicasting in Delay Tolerant Networks: A Social Network Perspective," *Proc. ACM MobiHoc*, 2009.
[10] F. Li, A. Srinivasan, and J. Wu, "Thwarting Blackhole Attacks in Distruption-Tolerant Networks Using Encounter Tickets," *Proc. IEEE INFOCOM*, 2009.
[11] Y. Ren, M.C. Chuah, J. Yang, and Y. Chen, "Detecting Wormhole Attacks in Delay Tolerant Networks," *IEEE Wireless Comm. Magazine*, vol. 17, no. 5, pp. 36-42, Oct. 2010.
[12] U. Shevade, H. Song, L. Qiu, and Y. Zhang, "Incentive-Aware Routing in DTNS," *Proc. IEEE Int'l Conf. Network Protocols (ICNP '08)*, 2008.
[13] Q. Li and G. Cao, "Mitigating Routing Misbehavior in Disruption Tolerant Networks," *IEEE Trans. Information Forensics and Security*, vol. 7, no. 2, pp. 664-675, Apr. 2012.

[14] H. Zhu, X. Lin, R. Lu, X.S. Shen, D. Xing, and Z. Cao, "An Opportunistic Batch Bundle Authentication Scheme for Energy Constrained DTNS," *Proc. IEEE INFOCOM*, 2010.

[15] B. Raghavan, K. Vishwanath, S. Ramabhadran, K. Yocum, and A. Snoeren, "Cloud Control with Distributed Rate Limiting," *Proc. ACM SIGCOMM*, 2007.

[16] F-SECURE, "F-Secure Malware Information Pages: Smsworm:-Symbos/Feak," http://www.f-secure.com/v-descs/smsworm symbos feak.shtml, 2012.

[17] N. Eagle and A. Pentland, "Reality Mining: Sensing Complex Social Systems," *Personal and Ubiquitous Computing*, vol. 10, no. 4, pp. 255-268, 2006.

[18] Q. Li, S. Zhu, and G. Cao, "Routing in Socially Selfish Delay Tolerant Networks," *Proc. IEEE INFOCOM*, 2010.

[19] T. Spyropoulos, K. Psounis, and C.S. Raghavendra, "Efficient Routing in Intermittently Connected Mobile Networks: The Multiple-Copy Case," *IEEE/ACM Trans. Networking*, vol. 16, no. 1, pp. 77-90, Feb. 2008.

[20] A. Lindgren, A. Doria, and O. Schelen, "Probabilistic Routing in Intermittently Connected Networks," *ACM SIGMOBILE Mobile Computing and Comm. Rev.*, vol. 7, no. 3, pp. 19-20, 2003.

[21] W. Gao and G. Cao, "On Exploiting Transient Contact Patterns for Data Forwarding in Delay Tolerant Networks," *Proc. IEEE 18th Int'l Conf. Networks Protocols (ICNP)*, 2010.

[22] J. Burgess, G.D. Bissias, M. Corner, and B.N. Levine, "Surviving Attacks on Disruption-Tolerant Networks without Authentication," *Proc. ACM MobiHoc*, 2007.

[23] S.C. Nelson, M. Bakht, and R. Kravets, "Encounter-Based Routing in Dtns," *Proc. IEEE INFOCOM*, pp. 846-854, 2009.

[24] T. Spyropoulos, K. Psounis, and C. Raghavendra, "Spray and Wait: An Efficient Routing Scheme for Intermittently Connected Mobile Networks," *Proc. ACM SIGCOMM*, pp. 252-259, 2005.

[25] B. Chen and C. Choon, "Mobicent: A Credit-Based Incentive System for Disruption Tolerant Network," *Proc. IEEE INFOCOM*, 2010.

[26] C. Gentry and A. Silverberg, "Hierarchical Id-Based Cryptography," *Proc. Int'l Conf. Theory and Application of Cryptography and Information Security EUROCRYPT*, 2002.

[27] A. Seth, D. Kroeker, M. Zaharia, S. Guo, and S. Keshav, "Lowcost Communication for Rural Internet Kiosks Using Mechanical Backhaul," *Proc. ACM Mobicom*, 2006.

[28] A. Vahdat and D. Becker, "Epidemic Routing for Partially Connected Ad Hoc Networks," Technical Report CS-200006, Duke Univ., 2000.

[29] R. Merkle, "Protocols for Public Key Cryptosystems," *Proc. IEEE Symp. Security and Privacy*, 1980.

[30] R. Groenevelt, "Stochastic Models in Mobile Ad Hoc Networks," technical report, Univ. of Nice, Sophia Antipolis, INRIA, 2006.

[31] A. Chaintreau, A. Mtibaa, L. Massoulie, and C. Diot, "The Diameter of Opportunistic Mobile Networks," *Proc. ACM CoNEXT Conf.*, 2007.

[32] P. Hui, J. Crowcroft, and E. Yoneki, "Bubble RAP: Social-Based Forwarding in Delay Tolerant Networks," *Proc. MobiHoc*, pp. 241-250, 2008.

[33] B. Parno, A. Perrig, and V. Gligor, "Distributed Detection of Node Replication Attacks in Sensor Networks," *Proc. IEEE Symp. Security and Privacy*, 2005.

[34] S. Zhong, J. Chen, and Y.R. Yang, "Sprite: A Simple, Cheat-Proof, Credit-Based System for Mobile Ad-Hoc Networks," *Proc. IEEE INFOCOM*, vol. 3, pp. 1987-1997, 2003.

[35] V. Natarajan, Y. Yang, and S. Zhu, "Resource-Misuse Attack Detection in Delay-Tolerant Networks," *Proc. Int'l Performance Computing and Comm. Conf. (IPCCC)*, 2011.

[36] Q. Li, W. Gao, S. Zhu, and G. Cao, "A Routing Protocol for Socially Selfish Delay Tolerant Networks," *Ad Hoc Networks*, vol. 10, no. 8, November 2012.

[37] W. Gao, G. Cao, M. Srivatsa, and A. Iyengar, "Distributed Maintenance of Cache Freshness in Opportunistic Mobile Networks," *IEEE ICDCS*, 2012.

[38] Z. Zhu and G. Cao, "Applaus: A Privacy-Preserving Location Proof Updating System for Location-Based Services," *IEEE INFOCOM*, 2011.

[39] Q. Li and G. Cao, "Efficient and Privacy-Preserving Data Aggregation in Mobile Sensing," *Proc. IEEE Int'l Conf. Network Protocols (ICNP '08)*, 2012.

Qinghua Li received the BE degree from Xian Jiaotong University, China, and the MS degree from Tsinghua University, China, in 2004 and 2007, respectively. He is currently a PhD candidate in the Department of Computer Science and Engineering at the Pennsylvania State University. His research interests include wireless networks and network security. He is a student member of the IEEE.

Wei Gao received the BE degree in Electrical Engineering from University of Science and Technology of China in 2005, and the PhD degree in computer science from the Pennsylvania State University in 2012. He is currently an assistant professor in the Department of Electrical Engineering and Computer Science at the University of Tennessee, Knoxville. His research interests include wireless and mobile network systems, mobile social networks, cyber-physical systems, pervasive and mobile computing. He is a member of the IEEE.

Sencun Zhu received the BS degree in precision instruments from Tsinghua University, Beijing, China, in 1996 and the MS degree in signal processing from the University of Science and Technology of China, Graduate School at Beijing, in 1999, and the PhD degree in information technology from George Mason University in 2004. His research interests include network and systems security and software security. Currently he is working on issues related to ad hoc and sensor network security, cellphone security, and security and privacy in online social networks. His research is funded by NSF and ARL. He is a member of the Networking and Security Research Center and is also affiliated with the Cyber Security Lab.

Guohong Cao received the BS degree in computer science from Xian Jiaotong University and received the PhD degree in computer science from the Ohio State University in 1999. Since then, he has been with the Department of Computer Science and Engineering at the Pennsylvania State University, where he is currently a professor. He has published more than 150 papers in the areas of wireless networks, wireless security, vehicular networks, wireless sensor networks, cache management, and distributed fault tolerant computing. He has served on the editorial board of *IEEE Transactions on Mobile Computing*, *IEEE Transactions on Wireless Communications*, *IEEE Transactions on Vehicular Technology*, and has served on the organizing and technical program committees of many conferences, including the TPC Chair/Cochair of IEEE SRDS 2009, MASS 2010, and INFOCOM 2013. He was a recipient of the NSF CAREER award in 2001. He is a Fellow of the IEEE.