Making DTNs Robust Against Spoofing Attacks with Localized Countermeasures

Md Yusuf Sarwar Uddin, Ahmed Khurshid, Hee Dong Jung, Carl Gunter, Matthew Caesar, Tarek Abdelzaher

Department of Computer Science
University of Illinois at Urbana-Champaign
{mduddin2, khurshi1, jung42, cgunter, caesar, zaher}@illinois.edu

***Abstract*—In this paper, we propose countermeasures to mitigate damage caused by spoofing attacks in Delay-Tolerant Networks (DTNs). In our model, an attacker spoofs someone else's address (the victim's) to absorb packets from the network intended for that victim. Address spoofing is arguably a very severe attack in DTNs, compared to other known attacks, such as dropping packets. Without a Public Key Infrastructure in DTNs, providing protection against this attack is challenging. We propose SPREAD (countermeasure against SPoofing by REplica ADjustment), a solution that assesses evidence of spoofing and offers countermeasures designed for quota-based multi-copy routing protocols. Our solution relies on reducing the weight of packet copies, charged to the routing quota, when these packets are given to a node suspected of spoofing. The weight reduction increases as spoofing evidence mounts against a node. The approach is designed to probabilistically maintain the same number of packet copies in the network as would be the case in the absence of attacks, despite the actual occurrence of spoofing. We show that SPREAD makes DTNs robust against spoofing attacks, does not overburden the network, and limits the overall overhead within a certain bound[1].**

I. Introduction

Delay-Tolerant Networks (DTNs) [1] experience intermittent connectivity due to various kinds of disruptions such as unavailability of links (e.g., links to low-earth orbit satellites), short transmission range of sparsely located mobile nodes (e.g., disaster-response networks), and limited energy resources (e.g., duty-cycled sensor networks). In DTNs, packets are not transferred along a connected multi-hop path. Instead, nodes store packets in their buffers until the next link becomes available (or, in some cases, physically carry packets onward) and transfer them to the next node when a transfer opportunity arises. The wait between node encounters requires a larger amount of time for packets to reach their destinations, requiring services constructed atop this infrastructure to be delay-tolerant.

Like other distributed communication networks, DTNs may suffer from malicious behavior of participants. Unfortunately, DTNs are often deployed in settings where network operators lack the ability to tightly monitor and repair nodes from anomalous behavior, making it difficult to detect and localize attacks. Nevertheless, DTNs are used for scientific, military, and industrial applications that place high demands on correctness of operation. There are several kinds of attacks that are possible in DTNs, such as packet dropping, flooding with unnecessary spurious packets, corrupting routing state, and counterfeiting network acknowledgments [2], [3]. In [3], the authors evaluate the robustness of DTN protocols without an authentication system. They show that network-wide delivery of packets can be reduced by a set of attacks including packet drops and routing state pollution. Another paper [4] shows a similar set of results for a particular routing protocol, MaxProp, in the presence of attackers that tamper with routing metrics. None of these efforts, however, provide countermeasures against those attacks.

Many of the above attacks can be addressed by conventional solutions, such as encryption and authentication. One unique challenge is that authentication is really hard to achieve in DTNs, due to the challenges of deploying Public Key Infrastructure (PKI) based solutions with trusted Certificate Authorities (CAs) [2], [3], [4]. This makes DTNs prone to yet another severe attack; namely, the *spoofing* attack. In DTNs, nodes do not usually remove packets from their buffers when they forward them. Instead, they retain such packets for further replication (making multiple copies of each packet). A packet is buffered until either the relay node delivers it to its ultimate destination or the packet's TTL (time to live) expires. After delivery, the packet is removed from the buffers. Exploiting this method of operation, an attacker hiding its real identity can claim someone else's identity (the victim's) and explicitly solicit packets from other nodes that are destined for the victim. Once it receives these packets, the attacker simply drops them. The sender also removes them, thinking they have been delivered to their intended destination. Spoofing is arguably a severe attack, because it not only results in removal of packets by the attackers themselves, but also causes other relay nodes (possibly non-attackers) to remove packets from their buffers as they think they delivered those packets, whereas they actually did not. Without authentication, combating this type of attack is very challenging.

We propose a localized solution in order to reduce the damage caused by the attackers without using any network-wide authentication procedure. We do not necessarily stop nodes from spoofing, neither do we detect nor isolate spoofers so that honest nodes can avoid them while forwarding data

[1]In the Proceedings of the 8th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2011, Salt Lake City, Utah, USA, June 27–30, 2011

packets. Instead, we attempt to make the whole network *robust* enough so that even though there could be spoofing attacks, the end performance of the network would not degrade much. A key element of the approach is that it is *adaptive* to the attack as explored in several DoS mitigation techniques [5], [6] and we investigate a new one. In our approach, nodes continuously assess the level of attacks in the network and adaptively create more replicas of packets. The adaptive nature of the solution ensures that the solution incurs nearly zero cost when there is no attacker. Moreover, the solution is localized in the sense that here nodes act independently without any coordination among themselves.

To this end, we propose SPREAD (countermeasure against SPoofing by REplica ADjustment), a *localized* countermeasure against spoofing attacks in DTNs. To the best of our knowledge, we are the first to propose a countermeasure for spoofing attacks in DTNs. SPREAD uses the basic DTN operation, replication of packets, to produce a few more copies of packets to reduce the ultimate effect caused by the attackers. SPREAD allows nodes to *locally* assess spoofing evidence against addresses and is designed for quota-based multi-copy routing protocols that limit, by setting a quota, the maximum number of replicas per packet. Once spoofing evidence is detected, SPREAD spreads more copies of affected packets into the network, by reducing the amount charged by each replica to the replication quota. More copies are injected as spoofing evidence mounts against an address. The accounting scheme ensures that SPREAD probabilistically maintains the same number of packet copies in the network in the presence of attackers, as would be the case in the absence of attacks, despite the actual occurrence of spoofing. Thus, SPREAD does not overburden the network and limits the overall overhead within a certain bound.

We simulate spoofing attacks and our proposed countermeasure in ns-2. We made necessary extensions to ns-2 to support DTN environments. Simulation results show that packet delivery ratio decreases significantly in the presence of attackers and then improves when SPREAD is applied. We also measure the overhead caused by SPREAD in terms of the number of copies per packet and show that overhead remains within a certain bound.

II. Background and Related Work

There are several DTN models available, such as IPN (InterPlaNetary Internet) [7], DakNet [8], and DieselNet [9]. Examples of DTNs range from networks of public buses in a city to emergency response networks comprising relief vehicles and rescue workers deployed after a large-scale disaster. In this type of network, nodes are sparse and mobile. They physically carry packets around and occasionally "meet" (come within the radio range of) other nodes to exchange packets with them. Since such node encounters are less predictable and there are generally no guaranteed sequences for forwarding packets (akin to traditional paths in connected networks), traditional single-copy routing protocols are not usually suitable in DTNs. Most routing protocols proposed for DTNs allow multiple copies of a given packet to be created in order to increase the possibility of delivering the packet to the ultimate destination, despite failures of some of the paths. These are called *multi-copy* routing protocols.

In DTNs, packet transfer decisions are made upon a "contact" (meeting between a pair of nodes). Based on the identity of the peer node, the transfer event can be one of two types; *delivery* or *replication*:

- **Delivery** occurs when a node meets the final destination of a packet. On delivery, packets are transferred to the destination and removed from the sender buffer.
- **Replication** occurs when transfers are made to peers other than the final destination. When replicated, packets may still reside in the sender's buffer, but with an updated state (specific to the routing protocol used).

DTN routing protocols can be broadly classified into two major classes: *flooding-based* and *quota-based*. Flooding-based schemes replicate packets as many times as they require, whereas quota-based schemes set an explicit limit on the number of replicas per packet. Flooding-based schemes include Epidemic routing [10], PROPHET [11], MaxProp [12] and Delegation routing [13]. Epidemic routing simply floods the network with packets, whereas PROPHET estimates delivery predictability to destinations using meeting history and makes regulated flooding. MaxProp computes delivery probability (called rank) for each destination and replicates packets in the order of their ranks. Delegation routing keeps states in packet headers in the form of probability of delivery and updates this probability upon replication.

Spray-and-Wait [14], Encounter-Based Routing (EBR) [15] and Inter-contact Routing [16] are examples of quota-based protocols. In such protocols, each packet contains a header field, called *replica count* or *quota*, that indicates how many times the packet can be replicated. On replication, the quota is split, such that the sender retains some (logical) copies of the packet and the peer receives the remaining copies. The packet is not sent repeatedly to generate the logical copies. Rather it is sent once and the replica count field is adjusted accordingly. A *split factor*, r, decides what fraction of copies is retained by the sender for a given contact. Binary Spray-and-Wait uses $r = \frac{1}{2}$ for all contacts, whereas, EBR determines r based on popularity of nodes and Inter-contact routing computes it from delay distributions along paths and delivery probabilities. Each packet starts with an initial replica count, called *initial quota*, and replicates itself until the replica count becomes 1. At that point, the packet cannot be replicated further, but is buffered until it can be delivered directly to the ultimate destination or removed from the buffer when its TTL expires.

There have been a few research efforts on securing DTNs. Work by Seth et al. [17] has shown that traditional mechanisms including a combination of PKI certificates issued by trusted third parties and Certificate Revocation Lists (CRLs) are not suitable for DTNs. They, however, do not discuss how the disconnectedness and opportunistic nature of communication can be exploited by malicious agents to disrupt packet flow between legitimate nodes. Other approaches addressing secu-

rity issues in DTNs include [18], [19], [20]. In this work, we try to make DTNs robust against spoofing attacks instead of securing them.

Burgess et al. [3] show that DTNs are more robust to adversaries than usual connected networks. They compare single-copy routing methods with multi-copy ones and show that multi-copy methods yield a higher delivery ratio in the presence of malicious nodes. They compare four routing algorithms (MaxProp and its three variants) against four attack models: dropping packets, flooding packets, route falsification and counterfeiting delivery ACKs. They also show that performance of DTNs degrades by a mere 15% even in the presence of 30% adversary nodes for a particular set of attack models. They do not provide any countermeasure against those attacks. In this paper, we consider spoofing attacks and provide a countermeasure against the attack.

III. Attack/Threat Models

We consider spoofing attacks in DTNs, where attackers act maliciously by identity forgery. Without PKI, identity protection is hard in DTNs. An alternative could be to distribute a common public key *apriori* to all nodes. Some DTNs pose this to be infeasible. Consider the aftermath of a disaster. A set of entities arrive from different organizations; community fire-fighters, rescue workers from the Red-Cross, volunteers from distressed neighborhoods, and supply vehicles from relief centers. Establishing a common public key across all nodes would require global coordination that may not be feasible considering the immediate nature of deployment. Moreover, any node can be compromised with all of its credentials and can operate as an attacker. A few malicious users can deliberately act as spoofers, claiming identity of, say fire-fighters or volunteers, to disrupt rescue services. Beyond rescue operations, DTNs may arise in military scenarios, where attackers may be motivated to intercept packets and disrupt communication.

A spoofing attack is executed in the following way. An attacker learns addresses from the network. The detail of attackers discovering addresses may depend on node discovery protocols used by the network, and is not our concern. While initiating a session with another node when they are in contact, an attacker passes an address other than its real address. Consequently, the other peer (which may be an honest one) delivers the packets destined to this spoofed address and removes them from its buffer. Unless those packets have been sufficiently replicated earlier, their delivery probability to the actual destination would thus be reduced.

In each meeting, a spoofer can choose whatever address it likes. It is, however, reasonable to assume that attackers will not choose an address outside the network that does not belong to an existing node. Obviously, spoofing a non-existing address will not lead to interception of valid packets originating in the network and will not serve the attack intention. There can be several forms of spoofing attacks:

- *Single-node spoofing:* A set of spoofers all claim a single address intercepting all packets destined to that node, aiming to prevent the rest of the network from sending packets to the victim node. Here, attackers target a single node causing the classical Denial of Service (DoS) attack on the victim.
- *Peer-wise spoofing:* Attackers choose different addresses while meeting different peers, but consistently claim the same spoofed address when meeting the same peer. Since attackers need to remember which address they claimed to which peer, we call this a *stateful attack*, as discussed later on.
- *Random spoofing:* Spoofers can claim any random address they know of. This would intercept packets from random destinations, and network-wide delivery of packet will degrade evenly.

The packets for which a spoofer is the claimed destination are absorbed and taken off from the network by the attacker. Transit packets, intended for other addresses not spoofed by the attacker, can also be dropped. The attacker can do other things to transit packets as well, such as corrupting packet headers or putting garbled content in packet's body. It can alternatively forward them intact. In designing our countermeasure, we assume that transit packets forwarded to an attacker are lost too.

On the surface, address spoofing may resemble Sybil attacks. Sybil attackers usually produce multiple identities so that a single node can multiply its presence in the system. This is usually done for biasing results achieved via a majority consensus or to confuse a reputation system. A Sybil instance of a node prefers to appear as a new node so that the system recognizes one more member. In contrast, in spoofing attacks, an attacker pretends to be some other existing node instead of claiming to be a new one. They want to duplicate identities in place of making newer identities.

IV. SPREAD Mechanism

One possible protection mechanism against spoofing attacks is to identify attackers and avoid them. As we mentioned earlier, due to the lack of PKI, identity protection is not possible in DTNs. Instead, we increase robustness of the network against spoofing attacks and let the level of robustness depend on an individual node's own experience of spoofing evidence. In contrast to static wireless networks, we exploit two unique features of DTNs in designing our countermeasure: multi-copy routing and diversity of encounters. Multiple copies of the same packet allow several ways of delivering packets to destinations, even when a few of them are consumed by attackers. Mobility of nodes causes diversity of encounters that allows a given node to meet several other nodes and not remain surrounded by attackers all the time. This allows nodes to replicate their packets bypassing attackers, as long as they are able to keep enough copies per packet.

We propose SPREAD, a localized countermeasure against spoofing attacks. In SPREAD, nodes try to assess the presence of attackers in the network. The scheme focuses more on "assessing" evidence in support of spoofing, instead of "detecting" an individual spoofer. A probabilistic assessment

is performed of whether a particular address is spoofed, given prior evidence. While replicating or delivering packets to an address that may be spoofed, an honest node takes into consideration the fact that the other end could be an attacker and that forwarded packets might therefore be lost. Accordingly, it may choose not to remove packets from its buffer upon delivery or may create more copies of packets upon replication. These two actions improve packet delivery ratio in the presence of attackers. The challenge is to use these actions in moderation, without overburdening the network with too many replicas. SPREAD does not overburden the network and keeps the expected total number of replicas within a certain bound. In the following, we describe various components of SPREAD.

A. Assessing spoofing evidence

The spoofing assessment is done as follows. Every node X generates a unique secret (long bit string) for each peer address Y it encounters. This bit string is called a *token*, $T_X(Y)$. $T_X(Y)$ is computed from some hash function on ID of Y and a private string of X, P_X, i.e., $T_X(Y) = H_X(Y, P_X)$. Nodes exchange these tokens when they first meet. In subsequent meetings between the same pair of nodes, each of them reproduces the token that it received from the other earlier. Hence, when X sees again some node who claims it has address Y, X expects Y to reproduce the token $T_X(Y)$ that X sent to Y earlier. If a different token is returned instead (or an empty token is returned), we say that there is a token *mismatch*. A mismatch indicates that the address Y is claimed by two (or more) different nodes, indicating the presence of spoofers. In the absence of attackers, a node should experience zero token mismatches after the initial token exchange with each peer address.

Let $cn^X(Y)$ be the mismatch count for an address Y computed at a given node X, and $p_s^X(Y)$ be the probability that a node claiming address Y is a spoofer, as assessed by node X. The mismatch count may not necessarily reflect the accurate number of spoofers. It may be the case that a single attacker does not care about storing tokens and reproduces random tokens at each meeting, thereby artificially increasing the count. This is fine, however, because the pessimistic estimate of the number of spoofers will simply lead to increasing the number of packet replicas sent to that address, which has a beneficial effect on delivery ratio. For simplicity, we use this $cn^X(Y)$ as X's estimate of the number of potential spoofers who claimed address Y. Assuming uniformity of contacts, the probability that a node claiming address Y is a spoofer is thus estimated at X by:

$$p_s^X(Y) = 1 - \frac{1}{cn^X(Y) + 1} \tag{1}$$

Each node maintains a mismatch counter for each node it encounters. We simply use $p_s(Y)$ when node X that computes the spoofing probability is obvious from the context. Given a certain contact, i.e., meeting of two nodes, we even drop Y to simply write p_s, which denotes the probability that the other end's address of the current contact has been spoofed.

B. Replica adjustment by SPREAD

In DTNs, a node takes packet transfer decisions when it meets another node. Upon a contact, the node computes p_s for the peer and makes delivery and replication decisions for its buffered packets accordingly. Below we describe the actions taken by the node for an arbitrary packet P.

On delivery: When P is delivered to the peer, there is a p_s probability that the peer is a spoofer. Let α be the fraction of replicas remaining out of the initial quota. Note that, α starts with 1, and becomes 0 when replicas are exhausted and no further replication is allowed. A higher α value implies that a good number of copies are subject to be lost if P is given to an attacker. Hence, a node will retain a copy of P with higher probability when α is high. A node will also retain P with higher probability when p_s is high. Hence, upon delivery, the delivered packet is removed from the sender's buffer with the probability:

$$P\{\text{remove } P| \text{ delivery}\} = (1 - \alpha)(1 - p_s) \tag{2}$$

The above probability becomes 0 if $\alpha = 1$, implying that the very first copy (with $\alpha = 1$) that has not generated any copy yet would not be removed when delivered to a peer that is suspected of being spoofed.

On replication: In quota-based replication schemes, the sender adjusts the replica count of the packet based on its split factor for the current contact. Let the current replica count of a packet be k and r be the split factor. In these schemes, the sender keeps $\lceil rk \rceil$ copies and the peer gets $\lfloor (1 - r)k \rfloor$ copies. So, a total of k copies are maintained. In case the peer is an attacker (who eventually drops or corrupts the received packets), the fraction of copies replicated to the peer will be lost. In suspicion of possible loss of copies, the sender creates more replicas per packet, yet tries to keep the average number of replicas bounded by k. We call this the "k-copy invariant" principle.

Definition (k-copy invariant) Irrespective of whether the peer is an attacker or not, an honest sender raises the replica count of packet P in such a way that the total expected number of replicas for P remains the same as for the non-attacker case.

Let γ be the *raise factor* by which the replica count of packet P is multiplied, before replicating to the peer. That means, replica count becomes γk and then, the usual quota-based protocol is used to split the copies between the sender and the peer. If the peer is an honest node (with probability $1 - p_s$), a total of k copies are there. If the peer is a spoofer (with probability p_s), only the sender's copies remain. Due to the *k-copy invariant*, the expected number of copies should remain k, as follows:

$$\begin{aligned} E[K] &= (1 - p_s) \times \gamma k + p_s \times (r\gamma k) \\ \Rightarrow \gamma &= \frac{1}{1 - p_s(1 - r)} \end{aligned} \tag{3}$$

The last equation computes the raise factor for the current contact. Since $1 - r > 0$, we have $\gamma > 1$.

In the presence of attackers more replicas of packets are created when they are forwarded to a spoofed address. It

is important to remember that spoofing evidence is recorded per address. That means extra overhead due to creating more copies is only attributed to those addresses that are spoofed (with a few exceptions discussed in Section IV-E). This is particularly important in handling cases when a set of attackers spoof a certain target node leaving other nodes untouched. In that case, a certain address (the victim's) is affected and the countermeasure is only enforced while meeting with those nodes claiming that partciular address. This keeps overall overhead of the network limited, still providing higher delivery to the victim.

We argue that the assessment of spoofing evidence works due to two properties:

- An attacker cannot produce the token that an honest node generates for a particular peer, unless it has received that token from that honest node in the first place or it colludes with another attacker who knows the token. In evaluation, we show the sensitivity of our solution in the presence of colluders and it is shown that colluding does not degrade the results much.
- An attacker cannot avoid token mismatches at an honest node by spoofing an address that the honest node had already seen.

C. Bounded number of copies by SPREAD

Although SPREAD increases the number of copies, we can show that the expected number of total replicas per packet is bounded. The exact count of copies generated out of a packet actually depends on which contacts the packet passes through in the network and the corresponding p_s's and r's of those contacts. Since it is not possible to know of all these contacts for a particular packet, getting an exact count of copies is somewhat intractable. In order to obtain an approximate count of copies, in the following description we use the term p_s to denote an *average* spoofing probability of nodes instead of different probabilities for different peers at different nodes.

Theorem (Bounded number of replicas) *Out of replica quota k, SPREAD produces a bounded number of copies per packet and creates on average $\frac{k}{1-p_s}$ copies, where p_s is the average spoofing probability per peer address.*

Proof. Initially, all k copies are contained at the source in a single packet with replica count k. Let r be the common split factor and γ be the raise factor for all contacts. During replication the sender retains $r\gamma k$ copies, and the peer receives $(1-r)\gamma k$ copies. In the next contact, the sender retains $(r\gamma)^2 k$ copies and the other peer gets the remaining. Next, it keeps $(r\gamma)^3 k$ copies and so on. The replication continues until the sender ends up of having the last copy with replica count 1. The last copy cannot be replicated anymore, but can only be delivered. We demonstrate the process of copy creation by a tree, called a *spreading tree*, shown in Figure 1. Each internal node in the tree represents a contact and the leaves are the packet copies.

Due to the k-copy invariant principle, the expected sum of total copies at each level of the tree remains k. To prove that SPREAD creates a bounded number of copies, we need to show that the tree has a finite depth, yielding $(r\gamma)^l k = 1$ at some depth l. This can only happen if $r\gamma < 1$. We show that it holds.

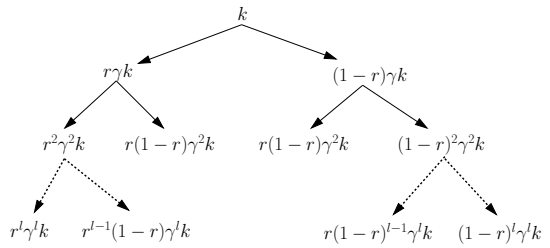


Fig. 1. A spreading tree (each internal node represents a contact)

$$r\gamma = \frac{r}{1-p_s(1-r)} = \frac{1}{p_s + \frac{1}{r}(1-p_s)} \tag{4}$$

Since $\frac{1}{r} > 1$ and $0 \leq p_s < 1$, hence $p_s + \frac{1}{r}(1-p_s) > 1$. Therefore, $r\gamma < 1$. Actually, we can also compute the depth l from $(r\gamma)^l k = 1$, that is, $l = \frac{\ln(k)}{-\ln(r\gamma)}$.

The spreading tree has k copies at its last level, all supposedly with replica count 1. These last copies may also create further copies because they are not always deleted from buffers as they are delivered to the destinations, but retained with probability p_s (i.e., removed with probability $1 - p_s$). Each successive delivery attempt effectively generates one more copy, if not removed in that attempt. Due to geometric distribution, each last copy is tried on an average $\frac{1}{1-p_s}$ times until it gets removed. Therefore, a total of $\frac{k}{1-p_s}$ copies are created. □

In practice, the number of copies would be quite smaller, because packets may be delivered before their replica count reaches 1, not generating successive copies afterward. It may seem that SPREAD creates an infinite number of copies as p_s approaches 1. Actually it does not, because packets would eventually be removed when their TTLs expire.

There is, however, a case when SPREAD creates a large number of copies where attackers only spoof addresses, but relay transit packets instead of dropping them. We know that in the absence of dropping, due to the raise factor, a total of γk copies are created at each contact. Copies are split between nodes upon a contact. We can compute the total number of copies as the sum of replica counts of leaf nodes of the spreading tree (Figure 1), which is $\sum_{i=1}^{l} \binom{l}{i} r^i (1-r)^{l-i} \gamma^l k = (r + (1-r))^l \gamma^l k = \gamma^l k$. Considering the last copy removal issue as before, a total of $\frac{\gamma^l}{1-p_s} k$ copies can be created. Since our usage of p_s's is an overarching simplification of the actual copy generation situation, these bounds hold only approximately in an average sense.

This really raises a concern. Huge number of replicas may cause congestion in the network. We can argue that packets passed through an attacker cannot be trusted anyway, so we need at least a few copies that are clean and SPREAD ensures that. But SPREAD cannot remove those extra packets by itself other than naturally dropping them when their TTLs expire. Moreover, it may appear that SPREAD costs near to flooding

as p_s goes up. Unlike flooding though, SPREAD does not burden network with replicas even if the network does not have any attackers. It does not do anything more than necessary. In evaluation, we show that overhead due to SPREAD is far less than flooding. To limit this extra overhead, we can put a cap on p_s so that replica count does not go beyond certain limit. We can even think of classifying packets into "important" and "ordinary" types so that most important packets are replicated more while leaving ordinary packets subject to attacks.

D. Stateless vs. Stateful attacks

Since a high mismatch count brings forth more replicas into the network, a question arises whether an attacker should remember the token it receives from a peer and spoof the same address when it meets that particular node. An attacker that does so may avoid further mismatches at the corresponding peer. This generates fewer replicas yielding a lower delivery ratio, consistently with the intent of the attack. This type of attack is called a *stateful* attack. This attack requires an attacker to remember the token it obtains from a node and the associated spoofed address. The other variant that simply chooses a random address to spoof and a random token can be regarded as *stateless* attack. In stateless spoofing, attackers may not remember previously received tokens. This causes repeated increment of token mismatch counts leading to a higher number of replicas.

E. Aspects of SPREAD

We describe a few important aspects of SPREAD as well as a few limitations along with associated possible remedies.

Underestimation of p_s: Suppose, node X meets only one node who claims an address Y and happens to be a spoofer, not the real Y. Equation 1 concludes that Y is honest by calculating $p_s(Y) = 0$. Again, it can happen that several nodes claim Y, but the real Y has never been met. In that case, $p_s(Y)$ should be 1, but it computes less than that. In either case, the ultimate packet delivery to Y by X is not affected if X never meets Y anyway. If node X ultimately meets Y, then the mismatch count is properly updated.

$p_s \geq 0.5$, if not 0: Due to Equation 1 when $cn > 0$. This means that when an address gets spoofed, there are at least two nodes involved; the honest node and a single attacker producing all spoofing evidence. So, spoofing probability can be at least 0.5.

Non-intrusiveness: This is a very strong feature of SPREAD. Without any attacker in the network (i.e., $p_s = 0$), SPREAD results in $\gamma = 1$ (Equation 3). That means it does not raise the replica count at all. This indicates that SPREAD is absolutely *non-intrusive* in that it does not cause additional overhead to the network unless there is any attacker. Extra overhead is incurred only when there are attackers.

False positives: A node stores only one token per address. Although it may receive multiple tokens from different nodes (possibly from spoofers), SPREAD stores only the most recent one. This may produce false positives. Suppose, node Y meets node X and receives a token $T_X(Y)$. Then, Y meets X', who also claims to be X, and overwrites $T_X(Y)$ with $T_{X'}(Y)$. Next, it meets the real X again, and reproduces the corresponding token which is $T_{X'}(Y)$. In this case, X detects a mismatch since $T_{X'}(Y)$ is different from $T_X(Y)$. It will increase the mismatch count for Y, whereas in fact the address Y is not spoofed. If Y meets X and X' alternately, the mismatch count for Y at node X would grow unboundedly. Figure 2 describes the situation. Occurrence of this kind of situation is however very rare, particularly the strict alternate meeting with X and X'. One solution to this problem could be to store all tokens received per address, and then reproduce the entire list so that the receiving node can verify whether its token is included.

Higher mismatch counts: Stateless spoofing does not reproduce tokens, thus increasing mismatch counts, which makes $p_s \rightarrow 1$. Then, $r\gamma$ becomes 1 (Equation 4) allowing the sender to retain $r\gamma k = k$, i.e., *all* copies, as if the current contact has not happened. This leads to a large number of packet copies because packets are now removed only when their TTL's are expired. A dropping policy or some congestion control schemes (e.g., Retiring Replicant [21]) can be applied to reduce copies.

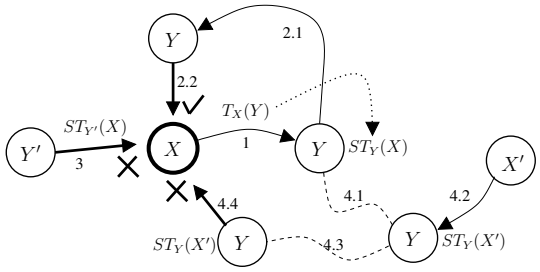


Fig. 2. A token distribution and reproduction scenario. 1. Node X meets node Y, passes its token, $T_X(Y)$, to Y, and Y stores the token as $ST_Y(X)$. 2. Y meets X again and reproduces the stored token $ST_Y(X)$. The reproduced token matches with $T_X(Y)$, hence no token-mismatch occurs (actually, $ST_Y(X) \neq T_X(Y)$ indicates a token mismatch). 3. X meets Y' (another node claiming itself Y) and Y' returns $ST_{Y'}(X)$ that does not match with $T_X(Y)$ and a token mismatch is detected at X (Y is spoofed). 4. Y meets X' (a spoofed X) and replaces its stored token by $ST_Y(X')$. Then, Y meets X again and gets its returned token mismatched at X (false positive).

F. Token management

Nodes generate tokens using a hash function (e.g., SHA-1) on an in-node secret key and the peer address for which the token is generated. Nodes maintain a peer token table (PTT) where they store the tokens received from other peers and the current mismatch count, indexed by peer address. Initially all the entries in PTT are empty. When two nodes meet, they first learn each other's address by exchanging HELLO messages. Then they extract the corresponding peer token from PTT (if available) and send it in the HELLO-REPLY message. Then, the token match is done. If the token does not match, the associated mismatch count is incremented and the token for that peer is sent by SET-TOKEN message. Note that nodes do not store the tokens they send to other nodes. They are easily regenerated whenever required.

Tokens are private to nodes. It is not possible for attackers to know or guess the tokens computed for other peers, unless

they can overhear or collude. DTNs are generally less subject to overhearing problems because of wide spatial separation among nodes. Yet the token can be passed encrypted by establishing a session key via the popular Diffie-Hellman protocol. Subsequent token exchange may not happen in plaintext either. When both parties have their tokens in place, they can challenge each other by sending a random integer and asking the other one to return the next integer hashed by the stored token. Both ends can easily verify whether the other end has the desired token or not. For simplicity of implementation, we assume that tokens are exchanged in plaintext by HELLO-REPLY and SET-TOKEN messages.

G. Implementing SPREAD

As a proof of concept, we instrument SPREAD onto the Spray-and-Wait routing protocol (which we refer to as Spray). Spray has $r = \frac{1}{2}$. So, $\gamma = \frac{2}{2-p_s}$. We prefer Spray because it does not maintain any routing state that can otherwise be corrupted by attackers. Spray is the simplest quota-based protocol and provides a bare benchmark to evaluate our countermeasure. We augment the basic Spray to incorporate SPREAD, as shown below.

Delivery (Contact c, Packet P)
Compute p_s for contact c
After successful transfer
  $\alpha = P$.replica-count/initial-quota
  if P.replica-count $= 1$ **then** $\alpha = 0$
  Generate a random value $rv \sim$ uniform$(0, 1)$
  if not the first meeting and $(rv < (1 - \alpha)(1 - p_s))$ **then**
    Remove P from buffer

Replication (Contact c, Packet P)
Compute p_s for contact c
$\gamma \leftarrow 2/(2 - p_s)$
P.replica-count $\leftarrow \lceil \gamma \times P$.replica-count$\rceil$
After successful transfer
  P.replica-count $\leftarrow P$.replica-count / 2

All the results presented in Section V are based on this augmented Spray protocol.

V. Performance Evaluation

We use ns-2 to evaluate the effects of spoofing attacks in a DTN and the countermeasures offered by SPREAD.

A. Simulation setup

We extend ns-2 to incorporate DTN-like functionality. Two key functions we need to implement are: (i) support for disrupted connections among different nodes and (ii) a node's ability to transfer packets to any other node in the network when they are in contact. In our implementation, every DTN node creates a separate communication agent for every other node in the network, and all these agents are statically connected pair-wise beforehand. When it is time for two nodes to communicate, the associated agent pair is simultaneously activated. A pre-generated connection pattern file containing activation/deactivation schedule (as experienced by nodes due to their mobility) is provided to simulate disruption. A more detailed description is available in our technical report [22].

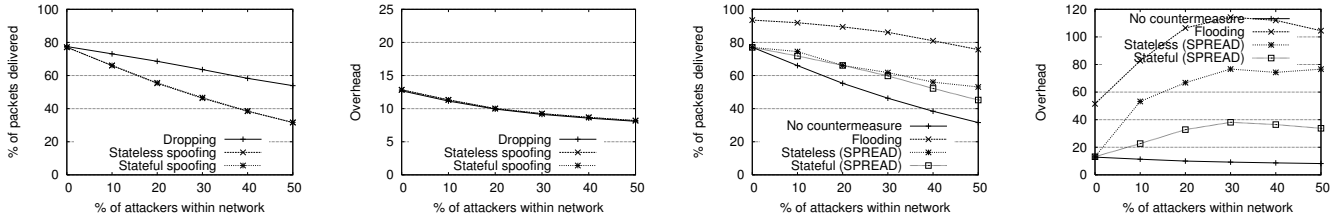
We use the ONE (Opportunistic Network Environment) simulator [23] to generate the connection pattern file used in our simulations. We use a city mobility model [23] with 3 kinds of nodes; 20 pedestrians (random movement within a confined area), 20 cars (random movement in the entire area) and 10 trams (4, 3 and 3 trams in 3 fixed cyclic routes respectively). The nodes run the *augmented* Spray protocol with an initial quota of 10. Packets are generated at a Poisson rate of 1 message per 5 minutes per node, and the simulation runs for 24 hours of simulated time.

B. Simulation results

We present our simulation results in terms of two major metrics: *packet delivery ratio* and *overhead*. Packet delivery ratio is the fraction of uniquely generated packets that have been delivered to their intended destinations. In the case of multiple copies reaching the destination, only the first delivery is counted. The overhead is computed as the total number of transmissions of all packets (i.e., copies) divided by the total number of unique packets transmitted at least once (we do not consider packets that have been generated but did not get transmission opportunities). During replication, each transmission generates a new copy. The final delivery needs one additional transmission. If there are k copies all with replica count 1, at most k additional transmissions can be made. To create k copies in the first place, it requires another $k - 1$ transmissions (the number of internal nodes in the spreading tree, Figure 1). So, for a replica quota k, at most $2k - 1$ transmissions can be made, resulting in that much overhead. Given the relationship between overhead and the number of packet copies, we will use these two terms interchangeably.

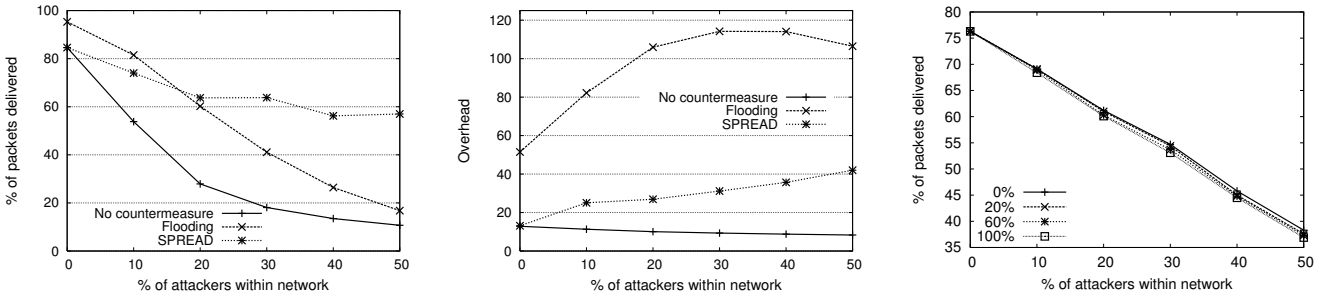
In Figure 3(a, b), we show the effects of attackers on delivery ratio and overhead without SPREAD. As we see in Figure 3(a), the delivery ratio declines as the number of attackers increases. To confirm the relative severity of spoofing attacks, we compare delivery ratio when attackers do not spoof addresses but only drop packets to that when attackers are spoofers. It reveals that the spoofing attack has a more adverse effect on delivery ratio than dropping. This is due to the fact that a spoofer, by claiming a false ID, not only grabs packets destined to other nodes, but also causes honest nodes to clear their buffers. Without SPREAD in action, both stateless and stateful spoofing result in nearly the same delivery ratio as well as a similar overhead (Figure 3(b)). Recall that a stateful spoofer remembers the address it needs to choose to avoid mismatches. Overhead remains the same for attackers that are spoofers or droppers. This is because spoofers also drop packets that they intercept. As more attackers drop packets, the replicas per packet gradually decline.

Figures 3(c, d) shows the same set of results when SPREAD is applied. We see that delivery ratio improves significantly with SPREAD (10% – 40%). We also observe that stateless spoofing results in a higher delivery ratio along with an increased overhead. This is because it causes more replicas to be created due to a larger number of mismatches. On the



(a) Delivery (w/o SPREAD) (b) Overhead (w/o SPREAD) (c) Delivery (with SPREAD) (d) Overhead (with SPREAD)

Fig. 3. Delivery ratio and overhead due to spoofing with and without SPREAD



(a) Single node attack: Delivery ratio (b) Single node attack: Overhead (c) Colluding attack: Delivery ratio

Fig. 4. (a, b) Delivery ratio and overhead when spoofers attack a single node, (c) Delivery ratio for colluding attackers

other hand, stateful spoofing results in a slightly lower delivery ratio as well as a lower overhead than that of stateless spoofing. Stateful spoofing seems to be a better attack than the stateless one in terms of delivery ratio, but a stateful attack allows countermeasure to be enforced with less expense.

In Figure 3(c), we demonstrate how SPREAD performs compared to flooding. We run the Epidemic flooding [10] protocol. Flooding does not follow any replication quota, but replicates packets as long as they are not delivered or expired. We observe that flooding offers a persistently higher delivery ratio ($> 80\%$), irrespective of the number of attackers, but at a very high overhead due to massive replication (Figure 3(d)). SPREAD's overhead against stateless spoofing is still very low compared to pure flooding. We note that a node does not send those packets that are already in the peer's buffer. As an attacker has an empty buffer, an honest node tends to send all packets to that peer, much higher than the case when there is no attacker. This causes overhead to rise, notably for flooding, compared to the zero attacker case, as more nodes become attackers. Beyond a certain fraction of attackers (after 30%), the dropping effect dominates over this extra replication. Then, overhead begins to decline.

Next, we consider the case when attackers target a single node and *all* attackers spoof the same victim's identity. The intention is to degrade the delivery ratio of the victim node to a very low value. Surprisingly, the attackers cannot achieve that when SPREAD is in action (Figure 4(a, b)). In the experiments, we choose a random node and allow all attackers to spoof that node's address. While calculating the delivery ratio, we only consider those packets originally destined to that address. We see that without any countermeasure, the delivery ratio declines drastically as more nodes spoof the victim's address. Even flooding does not improve upon this much, because flooding also removes packets from buffers once packets are delivered to a spoofer. SPREAD, however, retains a higher delivery ratio.

To calculate overhead for the above case, we consider the overall overhead instead of that of replicas only pertaining to the victim. This is because packets for other nodes are also affected (replica count is raised) while forwarded to a spoofed address. We observe that SPREAD keeps overall overhead very low, because it does not penalize all addresses with more replicas, but only does so for addresses that are spoofed.

We also show results when a certain fraction of attackers collude (Figure 4(c)). Colluding attackers are assumed to share their learned tokens through some "hidden" channels. Colluding attack seems like yet another stateful attack with at most one token mismatch accounted for multiple attackers. We observe that collusion does not degrade delivery ratio much compared to stateful attacks, even when 100% attackers collude. This is because an attacker cannot control the physical mobility of nodes to meet a certain node whose token it knows of to exploit identity forgery. Moreover, the probability of a node, to which the attacker meets, having packets destined to the spoofed address is quite low.

We now analyze internals of SPREAD. Recall that SPREAD counts token mismatches and replicates packets accordingly. Figure 5 depicts the relationship between the average mismatch count per address and the overhead exerted in the network thereby. We see that, as the number of attackers increases, nodes experience more mismatches that leads to higher overhead. Obviously, stateless spoofing produces more mismatches as well as more replicas than stateful spoofing.

In Figure 6, we show a summary of SPREAD's performance. In particular, we plot delivery ratio and overhead of SPREAD for zero and 20% attackers. As a baseline, the first

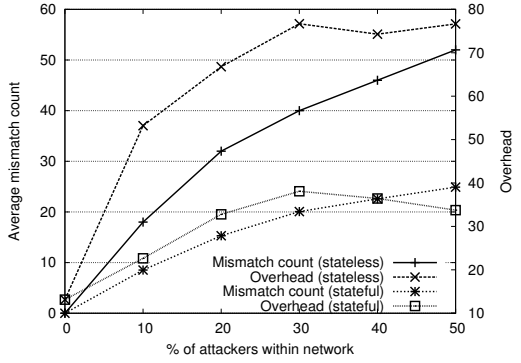


Fig. 5. Effect of mismatch count on overhead

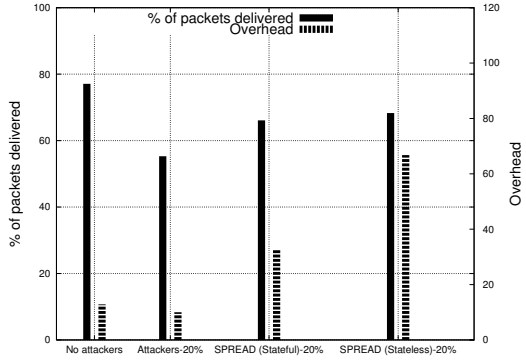


Fig. 6. Summary results of SPREAD with 0% and 20% attackers

two left bars are for the case without any countermeasure. Generally, the delivery ratio increases when SPREAD is applied. With SPREAD, stateless spoofing results in a better delivery ratio than stateful spoofing, but at the cost of higher overhead. Therefore, a stateless attacker may redirect its attack intention to "resource exhaustion" by inducing an excessive number of copies into the network rather than degrading the delivery ratio. This however generates higher delivery ratio. Resource exhaustion attack can be mitigated by rate-limiting transfer of packets on contact or putting caps on the highest possible number of replicas per packet. The following table shows different attack strategies for different intentions.

Countermeasure	Attack choice	Purpose of attack
NONE	Spoof	Delivery degradation
SPREAD	Stateless spoof	Resource exhaustion + Delivery degradation
SPREAD	Stateful spoof	Delivery degradation

VI. Conclusion and Future Work

In this paper, we consider spoofing attacks in DTNs. We show that spoofing reduces delivery ratio to a low value, either to a single node or network-wide. Our proposed countermeasure makes DTNs robust against spoofing attacks and significantly improves the delivery ratio with a slight, but bounded, increase in the number of packet copies.

In the future, we plan to investigate the possibility of designing trusted naming services and address discovery protocols for DTNs that can incorporate public keys with addresses. In that case, delivering packets to the wrong destinations can be prevented to some extent, subject to correct discovery of destination addresses and their public keys without a cerification authority. Another possible direction could be to understand whether having some infrastructure (may be a few nodes with trusted hardware) instead of a purely non-structured network could help in limiting spoofing attacks in DTNs.

Acknowledgment

This work was supported in part by Fulbright S&T Fellowship, ONR N00014-10-1-0172, NSF CNS 06-26825, DOE DE-0000097, HHS 90TR0003-01, NSF CNS 09-64392, NASA 09-VVFCS1-09-0010, NSF CNS 09-17218, NSF CNS 07-16421, NSF CNS 10-40391 and grants from the MacArthur Foundation and Lockheed Martin. The views expressed in this paper are those of the authors only.

References

[1] IRTF DTN Research Group, "http://www.dtnrg.org."
[2] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss, "Delay-tolerant networking architecture, RFC 4838," http://www.rfc-editor.org/rfc/rfc4838.txt.
[3] J. Burgess, G. D. Bissias, M. Corner, and B. N. Levine, "Surviving attacks on disruption-tolerant networks without authentication," in *Proc. of MobiHoc*, 2007.
[4] F. C. Choo, M. C. Chan, and E.-C. Chang, "Robustness of DTN against routing attacks," in *Proc. of COMSNETS*, 2010.
[5] M. Walfish, M. Vutukuru, H. Balakrishnan, D. Karger, and S. Shenker, "DDoS Defense by Offense," in *ACM SIGCOMM 2006*, September 2006.
[6] S. Khanna, S. S. Venkatesh, O. Fatemieh, F. Khan, and C. A. Gunter, "Adaptive selective verification," in *Proc. of INFOCOM*, 2008.
[7] InterPlanetary Internet, "www.ipnsig.org."
[8] A. Pentland, R. Fletcher, and A. Hasson, "DakNet: Rethinking connectivity in developing nations," *Computer*, vol. 37, no. 1, pp. 78–83, 2004.
[9] DieselNet, "http://prisms.cs.umass.edu/dome/umassdieselnet."
[10] A. Vahdat and D. Becker, "Epidemic routing for partially connected ad hoc networks," Department of Computer Science, Duke University, Tech. Rep. CS-2000-06, April 2000.
[11] A. Lindgren, A. Doria, and O. Schelén, "Probabilistic routing in intermittently connected networks," *Mobile Computing and Communications Review*, vol. 7, no. 3, pp. 19–20, July 2003.
[12] J. Burgess, B. Gallagher, D. Jensen, and B. Levine, "MaxProp: Routing for vehicle-based disruption-tolerant networks," in *Proc. of INFOCOM*, April 2006.
[13] V. Erramilli, M. Crovella, A. Chaintreau, and C. Diot, "Delegation forwarding," in *Proc. of MobiHoc*, 2007.
[14] T. Spyropoulos, K. Psounis, and C. Raghavendra, "Spray and Wait: An efficient routing scheme for intermittently connected mobile networks," in *Proc. of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking (WDTN '05)*, 2005.
[15] S. Nelson, M. Bakht, and R. Kravets, "Encounter-based routing in DTNs," in *Proc. of IEEE INFOCOM*, 2009.
[16] M. S. Uddin, H. Ahmadi, T. Abdelzaher, and R. Kravets, "A low-energy multicopy inter-contact routing protocol for disaster response networks," in *Proc. of IEEE SECON*, 2009.
[17] A. Seth and S. Keshav, "Practical security for disconnected nodes," *IEEE Workshop on Secure Network Protocols*, vol. 0, pp. 31–36, 2005.
[18] S. Symington, S. Farrell, and H. Weiss, "Bundle security protocol specification," May 2006.
[19] S. Farrell and V. Cahill, *Delay- and Disruption-Tolerant Networking*. Artech House, Inc., 2006.
[20] A. Kate, G. Zaverucha, and U. Hengartner, "Anonymity and security in delay tolerant networks," in *Proc. of SecureComm*, 2007.
[21] N. Thompson, S. Nelson, M. Bakht, T. Abdelzaher, and R. Kravets, "Retiring Replicants: Congestion control for intermittently-connected networks," in *Proc. of INFOCOM*, 2010.
[22] M. S. Uddin, A. Khurshid, H. D. Jung, and C. Gunter, "Denial in DTNs," UIUC, Tech. Rep., 2010.
[23] Opportunistic Network Environment simulator, "http://www.netlab.tkk.fi/ jo/dtn/index.html."