Practical Defenses Against Pollution Attacks in Intra-Flow Network Coding for Wireless Mesh Networks

Jing Dong
Department of Computer Science
Purdue University
West Lafayette, IN 47907
dongj@cs.purdue.edu

Reza Curtmola
Department of Computer Science
New Jersey Institute of Technology
Newark, NJ 07102
crix@njit.edu

Cristina Nita-Rotaru
Department of Computer Science
Purdue University
West Lafayette, IN 47907
crisn@cs.purdue.edu

ABSTRACT

Recent studies show that network coding can provide significant benefits to network protocols, such as increased throughput, reduced network congestion, higher reliability, and lower power consumption. The core principle of network coding is that intermediate nodes actively mix input packets to produce output packets. This mixing subjects network coding systems to a severe security threat, known as a *pollution attack*, where attacker nodes inject corrupted packets into the network. Corrupted packets propagate in an epidemic manner, depleting network resources and significantly decreasing throughput. Pollution attacks are particularly dangerous in wireless networks, where attackers can easily inject packets or compromise devices due to the increased network vulnerability.

In this paper, we address pollution attacks against network coding systems in wireless mesh networks. We demonstrate that previous solutions to the problem are impractical in wireless networks, incurring an unacceptably high degradation of throughput. We propose a lightweight scheme, DART, that uses time-based authentication in combination with random linear transformations to defend against pollution attacks. We further improve system performance and propose EDART, which enhances DART with an optimistic forwarding scheme. A detailed security analysis shows that the probability of a polluted packet passing our verification procedure is very low. Performance results using the well-known MORE protocol and realistic link quality measurements from the Roofnet experimental testbed show that our schemes improve system performance over 20 times compared to previous solutions.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communication*; C.2.m [**Computer-Communication Networks**]: Miscellaneous—*Security*

General Terms

Performance, Security

Keywords

Network coding, Pollution attacks, Network coding security, Wireless network security, Security

WiSec'09, March 16–18, 2009, Zurich, Switzerland.

1. INTRODUCTION

Network coding [1] introduces a new paradigm for network protocols. Recent research has demonstrated the advantages of network coding through practical systems such as COPE [2] and MORE [3] for wireless unicast and multicast, Avalanche [4] for peer-to-peer content distribution, protocols for peer-to-peer storage [5], and protocols for network monitoring and management [6–8]. Network coding has been shown to increase throughput [9–11], reduce network congestion [12], increase reliability [13, 14], and reduce power consumption [15–18], in unicast [3,19–22], multicast [3,23], and more general network configurations [24–27].

Unlike traditional routing, where intermediate nodes just forward input packets, in network coding, intermediate nodes actively mix (or code) input packets and forward the resulting coded packets. Original unencoded packets are usually referred to as *native packets* and packets formed from the mixing process are referred to as *coded packets*. The active mixing performed by intermediate nodes increases packet diversity in the network, resulting in fewer redundant transmissions and better use of network resources. However, the very nature of packet mixing also subjects network coding systems to a severe security threat, known as a *pollution attack*, where attackers inject corrupted packets into the network. Since intermediate nodes forward packets coded from their received packets, as long as at least one of the input packets is corrupted, all output packets forwarded by the node will be corrupted. This will further affect other nodes and result in the epidemic propagation of the attack in the network.

Wireless mesh networks are a promising technology for providing economical community-wide wireless access. Typically, a wireless mesh network consists of a set of stationary wireless routers that communicate via multi-hop wireless links. The broadcast nature of the wireless medium and the need for high throughput protocols make wireless mesh networks a prime environment for protocols based on network coding, and many such systems [2, 3, 22] have been developed. However, as recently shown in [28], the wireless environment makes the threat of pollution attacks particularly severe, since in wireless networks packets can be easily injected and bogus nodes can be easily deployed. Even in the presence of node authentication, such networks are vulnerable to insider attacks since wireless devices can be compromised and controlled by an adversary due to their increased susceptibility to theft and software vulnerabilities.

There are two general approaches for applying network coding to wireless mesh networks, *intra-flow* network coding and *inter-flow* network coding. Both approaches exploit the *broadcast advantage* and *opportunistic listening* in wireless networks to reduce transmissions and improve performance. However, these benefits are realized differently: Intra-flow network coding systems mix pack-

ets within a single flow, while inter-flow network coding systems mix packets across multiple flows.

In this paper, we focus on defense mechanisms against pollution attacks in intra-flow network coding systems for wireless mesh networks. In existing intra-flow coding systems [3, 23, 27], intermediate nodes do not decode received packets, but use them to generate new coded packets. However, intermediate nodes need to verify that each received coded packet is a valid combination of native packets from the source. As a result, traditional digital signature schemes cannot be used to defend against pollution attacks, because the brute force approach in which the source generates and disseminates signatures of all possible combinations of native packets has a prohibitive computation and communication cost, and thus it is not feasible.

Several solutions to address pollution attacks in intra-flow coding systems use special-crafted digital signatures [29–32] or hash functions [33, 34], which have homomorphic properties that allow intermediate nodes to verify the integrity of combined packets. While these are elegant approaches from a theoretical perspective, they are highly inefficient when applied in practice in wireless networks, even under benign conditions when no attacks take place. Non-cryptographic solutions have also been proposed [35–37]. These solutions either provide only a partial solution by detecting the attacks without any response mechanism [35], or add data redundancy at the source, resulting in throughput degradation proportionally to the bandwidth available to the attacker [36, 37].

We propose two practical schemes to address pollution attacks against intra-flow network coding in wireless mesh networks. Unlike previous work, our schemes do not require complex cryptographic functions and incur little overhead on the system, yet can effectively contain the impact of pollution attacks. To the best of our knowledge, this is the first paper which proposes practical defenses against pollution attacks in wireless networks and which demonstrates their effectiveness in a practical system. Our main contributions are:

- We demonstrate through both analysis and experiments that previous defenses against pollution attacks are impractical in wireless networks. In particular, we show that under a practical setting, previous cryptographic-based solutions [29–33] are able to achieve only less than 10% of typically available throughput.

- We design DART, a practical new defense scheme against pollution attacks. In DART, the source periodically disseminates random linear checksums for packets that are currently being forwarded in the network. Other nodes verify their received coded packets by checking the correctness of their checksums via efficient random linear transformations. The security of DART relies on *time asymmetry*, that is, a checksum is used to verify only those packets that are received *before* the checksum itself was created. This prevents an attacker that knows a checksum to subsequently generate corrupted packets that will pass our verification scheme, as the packets will be verified against another checksum that has not yet been created. DART uses pipelining to efficiently deliver multiple generations concurrently. Our analysis of the security of DART shows that under typical system settings, DART allows only 1 out of 65336 polluted packets passing a first hop neighbor of the attacker, and 1 out of over 4 billion polluted packets passing a second hop neighbor.

- We show how DART can be enhanced to perform optimistic forwarding of unverified packets in a controlled manner. The new scheme, EDART, improves network throughput and reduces delivery latency, while containing the scope of pollution attacks to a limited network region. Our analysis of EDART shows precise upper bounds on the impact of pollution attacks under the optimistic forwarding scheme.

- We validate the performance and the overhead of our schemes with extensive simulations using a well-known network coding system for wireless networks (MORE [3]) and realistic link quality measurements from the Roofnet [38] experimental testbed. Our results show that pollution attacks severely impact the network throughput; even a single attacker can reduce the throughput of most flows to zero. Our schemes effectively contain pollution attacks, achieving throughputs similar to a hypothetical ideal defense scheme. The schemes incur a bandwidth overhead of less than 2% of the system throughput and require approximately five digital signatures per second at the source.

Roadmap: Section 2 overviews related work. Section 3 presents our system and adversarial model, while Section 4 motivates the need for a new practical defense against pollution attacks. Sections 5 and 6 present our two schemes. Section 7 demonstrates the impact of the attacks and the effectiveness of our defense mechanisms through simulations. Finally, Section 8 concludes the paper.

2. RELATED WORK

Cryptographic approaches. In cryptographic approaches, the source uses cryptographic techniques to create and send additional verification information that allows nodes to verify the validity of coded packets. Polluted packets can then be filtered out by intermediate nodes. The proposed schemes rely on techniques such as homomorphic hash functions or homomorphic digital signatures. These schemes have high computational overhead, as each verification requires a large number of modular exponentiations. In addition, they require the verification information (*e.g.*, hashes or signatures) to be transmitted separately and reliably to all nodes in advance; this is difficult to achieve efficiently in wireless networks.

In hash-based schemes [33, 34], the source uses a homomorphic hash function to compute a hash of each native data packet and sends these hashes to intermediate nodes via an authenticated channel. The homomorphic property of the hash function allows nodes to compute the hash of a coded packet out of the hashes of native packets. The requirement for reliable communication is a strong assumption that limits the applicability of such schemes in wireless networks that have high error rates. The scheme proposed in [33] also has a high computational overhead. To overcome this limitation, [34] proposed probabilistic batch verification in conjunction with a cooperative detection mechanism. This scheme was proposed for and works reasonably well in peer-to-peer networks. However, it relies on fast and reliable dissemination of pollution alert messages. The scheme also relies on mask-based checksums that need to be sent individually to every node via different secret and reliable channels prior to the data transfer. Both of these are difficult to achieve in wireless networks, in which links have higher latency and error rate than in wired networks.

Schemes based on digital signatures [29–32] require reliable distribution of a new public key for every new file that is sent and the size of the public key is linear in the file size (the only exception is a recent scheme [39] which achieves constant-size public key, but uses expensive bilinear maps). The source uses specialized homomorphic signatures to send signatures that allow intermediate nodes to filter out polluted packets. These schemes have a high computational cost. To verify each packet, [29, 39] rely on expensive bilinear maps, while [30–32] require a large number of modular exponentiations. Although the schemes proposed in [30–32] allow *batch verification*, in which several packets are verified at once

to amortize the cost of verification, they have inherent limitations because they cannot achieve a suitable balance between computational overhead, network overhead, and packet delay. Ultimately, they result in low overall performance. In Sec. 4, we argue in detail that cryptographic approaches have high overhead even under benign conditions, making them impractical for use in a wireless network.

Information theoretic approaches. One information theoretic approach [35] relies on coding redundant information into packets, allowing receivers to efficiently detect the presence of polluted packets. The scheme provides only a partial solution, as it does not specify any mechanisms to recover from pollution attacks. Another approach [36] provides a distributed protocol to allow the receiver to recover native packets in the presence of pollution attacks. However, given that polluted packets are not filtered out, the throughput that can be achieved by the protocol is upper-bounded by the information-theoretic optimal rate of $C - z_O$, where C is the network capacity from the source to the receiver and z_O is the network capacity from the adversary to the receiver. Thus, if the attacker has a large bandwidth to the receiver, the useful throughput can rapidly degrade to 0. Unfortunately, there are many scenarios in wireless networks where the attacker has a large bandwidth to the receivers (*e.g.*, the attacker is located one hop away from the receiver, or multiple attackers are present), making the scheme not practical in wireless networks. In addition, due to the constrained bandwidth of the medium, there is a long term benefit in detecting the presence of the attacker and not allowing polluted packets to propagate in the network. [37] proposes to reduce the capacity of the attacker by only allowing nodes to broadcast at most once in the network. This model requires trusted nodes and differs vastly from practical systems for wireless networks, where each intermediate node in general forwards multiple coded packets.

Network error correction coding. Recent work [40–43] has developed a network error correction coding theory for detecting and correcting corrupted packets in network coding systems. In principle, the network error correction coding theory is parallel to classic coding theory for traditional communication networks, and also exhibits a fundamental trade-off between coding rate (bandwidth overhead of coding) and the error correction ability. Such schemes have limited error correcting ability and are inherently oriented toward network environments where errors only occur infrequently. In an adversarial wireless environment, the attackers are capable of injecting a large number of polluted packets that can easily overwhelm the error correction scheme and result in incorrect decoding.

3. SYSTEM AND ADVERSARIAL MODEL

3.1 System Model

We consider a general intra-flow network coding system where the network consists of a source s, multiple receivers $r_1, r_2, \ldots, r_k$, and other nodes, a subset of which are *forwarders* for packets. Receiver nodes may also act as forwarders. The source has a sequence of N packets which is divided into sub-sequences called *generations*. Each generation consists of n packets and it is disseminated independently to the receivers using network coding.

As required by network coding, each packet is divided into m codewords, each of which is regarded as an element in a finite field $\mathbb{F}_q$, where q is a positive power of a prime number. Each packet $\vec{p_i}$ can be viewed as an element in an m-dimensional vector space over the field $\mathbb{F}_q$, *i.e.*, as a column vector with m symbols:

$$\vec{p_i} = (p_{i1}, p_{i2}, \ldots, p_{im})^{\mathsf{T}}, p_{ij} \in \mathbb{F}_q.$$

A generation G consisting of n packets can be viewed as a matrix:

$$G = [\vec{p_1}, \vec{p_2}, \ldots, \vec{p_n}],$$

with each packet in the generation as a column in the matrix.

The source forms random linear combinations of uncoded packets $\vec{e} = \sum_{i=1}^{n} c_i \vec{p_i}$, where c_i is a random element in $\mathbb{F}_q$ and all algebraic operations are in $\mathbb{F}_q$. The source then forwards packets consisting of $(\vec{c}, \vec{e})$ in the network, where $\vec{c} = (c_1, c_2, \ldots, c_n)$. As in [3], we refer to uncoded packets as *native packets*, to $(\vec{c}, \vec{e})$ as *coded packets*, to $\vec{c}$ as the coded vector, and to $\vec{e}$ as the coded data. A forwarder node also forms new coded packets by computing linear combinations of the coded packets it has received and forwards them in the network. When a receiver has obtained n linearly independent coded packets, it can decode them to recover the native packets by solving a system of n linear equations. For consistency, all vectors used throughout the paper are column vectors.

This model is the general network coding framework proposed in [44] and fits all existing intra-flow network coding systems known to us, including [3, 22, 23, 45, 46]. We do not restrict the specific algorithm for selecting the subset of forwarder nodes, nor the packet coding and the forwarding scheme.

On the need of generations in network coding. To measure the communication overhead incurred by network coding, we use as a metric the *relative network overhead* $\rho = \frac{n}{m}$ (the ratio between the size of the code vector, which is overhead, and the size of the coded data, which is the useful data). The smaller the value of ρ, the less communication overhead incurred by network coding.

Given a fixed native packet size (*e.g.*, 1500B) and the size of field F_q, the value for m is fixed. Therefore, to ensure a small overhead ρ, the source has to encode the native packets in small chunks, i.e., *generations*, to ensure a small value for n. Otherwise, if the source applies network coding on the entire sequence of N packets (*i.e.*, treats the entire sequence as a single generation), then $\rho = \frac{N}{m}$. Clearly, for large files (*i.e.*, large N), this would result in large network overhead that would render network coding impractical. Instead, all practical systems designed for wireless networks [3, 22, 23] use network coding within generations of n packets (*e.g.*, $n = 32$). In general, when choosing the parameters for network coding, one needs to ensure $n \ll m$ in order to ensure a small communication overhead ρ.

The source advances through generations of packets based on a feedback mechanism which informs the source that all packets from a generation were received and decoded by receivers. We refer to a generation that is in transit from the source to the destination as an *active* generation. Depending on specific systems, multiple generations can be active at the same time.

3.2 Security and Adversarial Model

We assume the source is trusted. However, both the forwarders and receivers can be adversarial. They can either be bogus nodes introduced by the attacker or legitimate but compromised nodes. Adversarial nodes launch the pollution attack either by injecting bogus packets or by modifying their output packets to contain incorrect data. We say a packet $(\vec{c}, \vec{e})$ is a *polluted packet*, if the following equality does *not* hold:

$$\vec{e} = \sum_{i=1}^{n} c_i \vec{p_i}.$$

As in the well-known TESLA protocol [47, 48], we assume that clocks in the network are loosely synchronized and that each node knows the upper bound on the clock difference between the node and the source, denoted as Δ. Several mechanisms to securely achieve such loose clock synchronization are provided in [47, 49].

Other techniques proposed in [50] reduce the synchronization error to the order of microseconds.

We also assume the existence of an end-to-end message authentication scheme, such as traditional digital signature or message authentication code, that allows each receiver to efficiently verify the integrity of native packets after decoding.

We focus only on pollution attacks, which are a generic threat to all network coding systems. We do not consider attacks on the physical or MAC layer. We also do not consider packet dropping attacks, nor attacks that exploit design features of specific network coding systems, such as the selection of forwarder nodes. Defending against such attacks is complementary to our work.

4. LIMITATIONS OF PREVIOUS WORK

Approaches based on information theory and network error correction coding have severe limitations in wireless networks, as they assume limited bandwidth between the attacker and the receiver. However, in wireless networks, an attacker can easily have a large bandwidth to the receiver, for example, by injecting many corrupted packets, staying near the receiver node, or having multiple attacker nodes.

Cryptographic approaches propose to filter out polluted packets at the intermediate nodes by using homomorphic digital signatures [29–32] and homomorphic hashes [33,34]. Below we argue that the high computational cost of these schemes makes them impractical for wireless systems.

In the existing cryptographic-based schemes [29–34], verifying the validity of a coded packet requires $m + n$ modular exponentiations (typically using a 1024-bit modulus), where m is the number of symbols in a packet and n is the number of packets in a generation. Most schemes can reduce this cost to $\gamma = 1 + \frac{m}{n}$ exponentiations per packet by using *batch verification*, which enables nodes to verify a set of coded packets from the same generation at once. Note that batch verification cannot be performed across generations, since each generation requires a different set of parameters (*e.g.*, different public keys).

Even when batch verification is used, the computational cost is still too high for practical network coding systems. More precisely, since the relative network overhead due to network coding is $\rho = \frac{n}{m}$ and the computational overhead to verify a packet is $\gamma = 1 + \frac{m}{n}$, minimizing the computational cost and minimizing the relative network overhead are **two conflicting goals**; reducing one of them results in increasing the other. We now compute the maximum throughput τ achievable in such systems. We assume each node in the system is equipped with a 3.4 Ghz Pentium IV processor, which performs around 250 modular exponentiations per second[1] , and the packet size is 1500B. Therefore, the destination can verify $\frac{250}{\gamma}$ packets per second and the throughput is $\tau = (1 - \rho)\frac{250}{\gamma} \times 1500$ bytes per second, or equivalently,

$$\tau = (1 - \frac{n}{m})\frac{250 \times 1500}{1 + \frac{m}{n}},$$

which achieves the maximum value of 502kbps when $\frac{n}{m} = 0.41$. Therefore, even if the source and destination are direct neighbors and there is no attack, the maximum throughput achievable is 502kbps, regardless of the actual link bandwidth, which can be much larger, *e.g.*, 11Mbps. In practice, the source and destination are usually more than one hop away, in which case the achievable throughput is much lower. Our experiments (see Sec. 7.4) show that, under typical network settings, the achievable throughput is around 50kbps, which is only 4% of the throughput of the system without using the defense mechanism; this is a 96% throughput degradation even when no attacks take place.

Besides having a high computational cost, previously proposed cryptographic schemes require the source to disseminate new parameters for each generation (*e.g.*, a new public key per generation) with a size linear to the size of the generation. This further increases the overhead and reduces the throughput.

Previously proposed cryptographic schemes also have requirements that conflict with the parameters for practical network coding in wireless systems. A critical factor for the security of cryptographic schemes is the size of the field $\mathbb{F}_q$, which has to be large, *e.g.*, 20 or 32 bytes [29–32]. However, in all the practical network coding systems in wireless networks [3, 22, 23], the symbol size used is much smaller, usually one byte. This is because arithmetic operations over a field are extensively used and a small symbol size ensures that these operations are inexpensive. Furthermore, small symbols result in a large m value, which in turn reduces the relative network overhead.

5. THE DART SCHEME

Our first scheme, DART, uses checksums based on efficiently computable random linear transformations to allow each node to verify the validity of coded packets. The security of the scheme relies on *time asymmetry*, that is, a node verifies a coded packet only against a checksum that is generated after the coded packet itself was received. Each node uses only valid coded packets to form new coded packets for forwarding. Invalid packets are dropped after one hop, thus eliminating packet pollution. Our scheme can be applied on top of any network coding scheme that uses generations and has one or more active generations at a time. The time asymmetry of checksum verification in DART is close in spirit with TESLA[47], in which the sender delays disclosure of the key used to authenticate a packet.

We present our solution incrementally. First, we describe our scheme, focusing on one active generation. We present in detail the checksum generation and verification, showing how batch verification is performed for one generation. We then show how multiple generations can be pipelined in a network coding system to increase performance. Finally, we demonstrate the effectiveness of our scheme in filtering out polluted packets by analyzing the probability that an attacker bypasses our verification scheme.

5.1 Scheme Description

Let G be an active generation. The source *periodically* computes and disseminates a **random checksum** packet $(\mathsf{CHK}_s(G), s, t)$ for the generation, where $\mathsf{CHK}_s(G)$ is a random checksum for the packets in generation G, s is the random seed used to create the checksum, and t is the timestamp at the source when the checksum is created. The source ensures the authenticity of the checksum packet itself by digitally signing it[2].

Each forwarder node maintains two packet buffers, verified_set and unverified_set, that buffer the verified and unverified packets, respectively. Each node combines only packets in the verified_set to form new packets and forwards such packets as specified by the network coding system. On receiving a new coded packet, a node buffers the packet into unverified_set and records the corresponding receiving time.

Upon receiving a checksum packet $(\mathsf{CHK}_s(G), s, t)$, a forwarder node first verifies that it is not a duplicate and that it was sent by the source by checking its digital signature. If the checksum is

[1] Numbers were obtained using the OpenSSL library (version 0.9.8e).

[2] Alternatively, the source may use more efficient authentication mechanisms such as TESLA[47] or μTESLA[49].

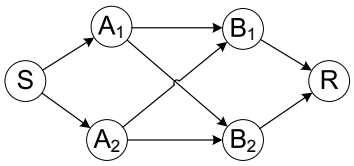


Figure 1: An example network for illustrating the process in DART. Node A_1, A_2, B_1, B_2 are the forwarder nodes for source S and destination R. The arrows indicate that packets can be heard over the link.

authentic, the node re-broadcasts it to its neighbors. It then uses the checksum to verify those packets in unverified_set that were received by that node before the checksum was created at the source (*i.e.*, packets whose receive time is smaller than the time $t - \Delta$, where Δ is the maximum time skew in the network). Valid packets are transferred from unverified_set to verified_set. Packets that do not pass the verification are discarded.

Checksum packets are not required to be delivered reliably: If a node fails to receive a checksum, it can verify its buffered packets upon the receipt of the next checksum. To reduce the overhead, we restrict the checksum to be flooded only among forwarder nodes.

When a receiver node receives enough linear independent coded packets that have passed the checksum verification, it decodes the packets to recover the native packets. It verifies the native packets using an end-to-end authentication scheme such as digital signature or message authentication code (MAC) before passing the packets to the upper layer protocol. The additional end-to-end authentication is to address the extremely rare occasion when some polluted packet pass our checksum verification at the receiver, which would otherwise cause incorrect packets to be delivered to the upper layer.

The key points of our approach are that checksums are very efficient to create and verify, as they are based on cheap algebraic operations, and that each node uses a checksum to verify only those packets that were received before the checksum itself was created. Therefore, although after obtaining a checksum an attacker is able to generate corrupted packets that match the known checksum, it cannot convince other nodes to accept them, as these packets will not be verified with the checksum known to the attacker, but with another random checksum generated by the source at a time after the packets are received.

Since coded packets are delayed at each hop for checksum verification, the number of checksums needed for a generation is at least as many as the number of hops from the source to the receiver. As checksums are released at fixed time intervals, the requirement of multiple checksums for a generation can result in a large delivery time for a generation, hence reducing throughput. The packet delivery time could be reduced by releasing checksums more often; however, this would increase the network overhead. We solve this dilemma by using pipelining across generations such that multiple generations are being transmitted concurrently. We describe pipelining in Sec 5.3.

Example. We illustrate DART with an example presented in Fig. 1. Without loss of generality, we assume that the source S transmits one active generation at a time and that nodes are perfectly time-synchronized. Let T denote the time interval at which the source periodically disseminates checksum packets.

At time 0, the source starts broadcasting coded packets for the current generation. Nodes A_1 and A_2 buffer their received coded packets in unverified_set and record their received time without forwarding them. At time T, S broadcasts a signed random checksum packet with timestamp T. Nodes A_1 and A_2 first forward the received checksum packet to the downstream nodes B_1 and B_2. Then they use the checksum to verify the packets in their unverified_set whose receive time is before time T, and transfer successfully verified packets to their verified_set. As nodes B_1 and B_2 do not have any packet to verify, the checksum packet is ignored at B_1 and B_2.

After packet verification, A_1 and A_2 start to forward new coded packets generated from their verified_set to B_1 and B_2, which buffer their received packets in their unverified_set without forwarding them.

At time $2T$, S broadcasts a new (different) random checksum for the generation, which is also forwarded by nodes A_1, A_2 to nodes B_1, B_2. Upon receiving the checksum packet, nodes A_1, A_2, B_1, B_2 use it to verify packets in their unverified_set that were received before time $2T$. Upon packet verification, nodes B_1, B_2 also start forwarding coded packets generated from their verified_set to the destination node R.

The above process continues until the destination node receives the entire generation of packets that are successfully verified, and sends an acknowledgment to the source, which advances the source to transmitting the next generation. The entire process then repeats for the delivery of the next generation.

5.2 Checksum Computation and Verification

We now describe in detail how checksums are generated and how individual coded packets are verified. We then show how to amortize the verification cost by verifying a set of packets at once.

As mentioned in the system model (Sec. 3.1), we denote the generation size used for network coding as n. Let $\vec{p_1}, \vec{p_2}, \ldots, \vec{p_n}$ be the packets to be transmitted in the current generation. We view each packet as an element in an m-dimensional vector space over a field $\mathbb{F}_q$, *i.e.*, as a column vector with m symbols:

$$\vec{p_i} = (p_{i1}, p_{i2}, \ldots, p_{im})^\mathsf{T}, p_{ij} \in \mathbb{F}_q.$$

We use a $m \times n$ matrix G to denote all packets in the generation:

$$G = [\vec{p_1}, \vec{p_2}, \ldots, \vec{p_n}].$$

Let $f : \{0,1\}^\kappa \times \{0,1\}^{\log_2(b)+\log_2(m)} \rightarrow \mathbb{F}_q$ be a pseudo-random function, where κ and b are security parameters (κ is the size of the key for f, whereas b controls the size of the checksum). We write $f_s(x)$ to denote f keyed with key s applied on input x.

Our checksum generation and verification are based on a random linear transformation applied on the packets in a generation.

Checksum creation. The source generates a random $b \times m$ matrix $H_s = [u_{i,j}]$ using the pseudo-random function f and a random κ-bit seed s, where $u_{i,j} = f_s(i||j)$. We define the checksum $\mathsf{CHK}_s(G)$ based on seed s for generation G as

$$\mathsf{CHK}_s(G) = H_s G.$$

Hence, $\mathsf{CHK}_s(G)$ is obtained by applying a random linear transformation H_s on the packets in G. Since H_s is a $b \times m$ matrix and G is a $m \times n$ matrix, the checksum $\mathsf{CHK}_s(G)$ is a $b \times n$ matrix. The source includes $(\mathsf{CHK}_s(G), s, t)$ in the checksum packet, where t is the timestamp at the source when the checksum is created, and then disseminates it in an authenticated manner.

Packet verification. Given an authentic checksum $(\mathsf{CHK}_s(G), s, t)$ for generation G, a node uses it to verify coded packets that are received before time $t - \Delta$, where Δ is the maximum time skew in the network. Given such a packet $(\vec{c}, \vec{e})$, a node checks its validity by checking if the following equation holds,

$$\mathsf{CHK}_s(G)\vec{c} = H_s\vec{e}, \quad (1)$$

where H_s is the random $b \times m$ matrix generated from seed s as described above.

If Eq. (1) holds, then the coded packet is deemed valid, otherwise, it is deemed invalid. To see the correctness of this check,

consider a valid packet $(\vec{c}, \vec{e})$, where $\vec{e} = \sum_{i=1}^{n} c_i \vec{p_i} = G\vec{c}$ and the checksum $(\mathsf{CHK}_s(G), s, t)$, where $\mathsf{CHK}_s(G) = H_s G$. Then,

$$\mathsf{CHK}_s(G)\vec{c} = (H_s G)\vec{c} = H_s(G\vec{c}) = H_s\vec{e}.$$

Batch verification. The above individual verification can be extended to efficiently verify a set of coded packets at once. Let

$$E = \{(\vec{c_1}, \vec{e_1}), \ldots, (\vec{c_l}, \vec{e_l})\}$$

be a set of l coded packets from a generation G where all packets are received before time $t - \Delta$. To verify E against a checksum $(\mathsf{CHK}_s(G), s, t)$ a node computes a random linear combination of the packets, $(\vec{c}, \vec{e}) = (\sum_{i=1}^{l} u_i\vec{c_i}, \sum_{i=1}^{l} u_i\vec{e_i})$, where the coefficients $u_1, u_2, \ldots, u_l$ are selected uniformly at random from $\mathbb{F}_q$. The node then verifies the combined packet $(\vec{c}, \vec{e})$ using the individual verification described above. A node can further reduce the false negative probability of the verification by repeating the procedure with different random coefficients.

If E passes the verification, then all l coded packets are regarded as valid. Otherwise, the invalid packets in the set can be identified efficiently using a technique similar to binary search.

Checksum overhead. The size of a checksum $(\mathsf{CHK}_s(G), s, t)$ is dominated by the size of $\mathsf{CHK}_s(G)$, which is a $b \times n$ matrix of elements in $\mathbb{F}_q$. Thus, its size is $bn \log_2 q$ bits. Compared to the total data size in a generation, the overhead is $(bn \log_2 q)/n(n + m) \log_2 q = b/(n + m)$. In a typical setting, $b = 2, n = 32, m = 1500$, the overhead is less than 0.1%. The computational overhead for checksum computation and verification is also comparable to generating a single coded packet in network coding, and is evaluated in Sec. 7.5.

5.3 Pipelining Across Generations

As discussed in Sec. 5.1, the basic DART scheme may reduce throughput due to the increased packet delivery time. A general approach to address this problem is using *pipelining*, in which transmissions are pipelined across generations and multiple generations are delivered concurrently. Several existing network coding systems [22, 23] already incorporate pipelining for performance purpose and DART can be applied directly to such systems without performance penalties. Next, we propose a generic mechanism for pipelining across generations in systems that do not perform pipelining natively, *e.g.*, MORE [3].

To pipeline packet transmission across generations, the source transmits n coded packets for each generation, moving to the next generation without waiting for acknowledgments. The source maintains a window of k active generations and cycles through these active generations, transmitting n coded packets for each generation. Whenever the source receives an acknowledgment for an active generation, that generation is considered successfully delivered and the source activates the next available generation of packets. Each checksum packet contains k checksum values, one for each active generation.

Selecting a large value for k assures that no link goes idle and the bandwidth resource is fully utilized. However, an overly large value for k increases the latency for delivering a generation, because the number of active generations that the source cycles through increases. To meet these two opposing requirements, the optimal k value should be the smallest value such that the bandwidth is fully utilized. We estimate the optimal k value as follows. Let d be the number of hops from the source to destination, and τ be the delay at each hop. τ consists of two components, the time between two checksum packets (t_1) and the clock synchronization error between the node and the source node (t_2), which is less than Δ. So the total delay from the source to the destination is $d\tau$. Let a be the time for transmitting n packets at the source, to ensure the source never idles, we need to have $k \geq \frac{d\tau}{a} = \frac{d(t_1+t_2)}{a}$. Assuming a relatively large clock synchronization error, that is $t_2 \gg t_1$, we have $k \geq \frac{dt_2}{a}$. Thus we can select $k = \frac{d\Delta}{a}$. Our experiments show that selecting $k = 5$ is sufficient.

A potential concern for pipelining is that the source needs to disseminate multiple checksums in one checksum packet, as there are multiple active generations simultaneously. Our experiments in Sec. 7 show that, due to the small size of checksums, the overall bandwidth overhead is still small.

5.4 Security Analysis

Below we discuss the security properties of DART by focusing on one generation. Pipelining across several active generations has no implication on the security analysis presented below as checksums are generation specific and packets for each generation are verified independently.

Recall that checksums are signed by the source, thus the attacker cannot inject forged checksums into the network. The only option left for the attacker is to generate corrupted packets that will match the checksum verification at honest nodes. The key point of our scheme that prevents this is the time asymmetry in checksum verification: A node uses a checksum to verify only packets that are received before the creation of the checksum. Therefore, unlike traditional hash functions where the attacker has a chance to find a collision because he has the hash value, in our scheme, the time asymmetry in checksum verification prevents the attacker from computing a suitable polluted packet that will pass the verification algorithm. At best, the attacker can randomly guess the upcoming checksum value, thus only having a small chance of success. We formalize the intuition for the security of our scheme as follows.

We say a coded packet is *safe* with respect to a checksum packet if the coded packet is created prior to the time the checksum packet is created at the source. We prove Lemma 1 and Theorems 1 and 2, where q is the field size used by network coding and b is the security parameter for the checksum generation as described in Sec.5.2. Due to space limitations, the proofs are presented in the full version of the paper [51].

LEMMA 1. *In DART, all packets that are verified against a checksum packet are safe with respect to that checksum packet.*

THEOREM 1. *Let $\mathsf{CHK} = (\mathsf{CHK}_s(G), s, t)$ be a checksum for a generation G, and let $(\vec{c}, \vec{e})$ be a polluted packet* (i.e., *$\vec{e} \neq G\vec{c}$). The probability that $(\vec{c}, \vec{e})$ successfully passes the packet verification for CHK at a node is at most $\frac{1}{q^b}$.*

THEOREM 2. *Let $\mathsf{CHK} = (\mathsf{CHK}_s(G), s, t)$ be a checksum for a generation G and let $E = \{(\vec{c_1}, \vec{e_1}), \ldots, (\vec{c_l}, \vec{e_l})\}$ be a set containing polluted packets. The probability that E successfully passes w independent batch verifications for CHK at a node is at most $\frac{1}{q^w} + \frac{1}{q^b}$.*

Note that the checksum verification algorithm does not have false positives. Thus, a packet can be verified against multiple checksums to further reduce the false negative probability, as long as the packet is safe with respect to the checksums. The failure of any checksum verification indicates that the packet is corrupted. However, our experimental results (Sec. 7) show that verifying each packet with only one checksum is already sufficient. Also note that since each checksum is generated independently at random,

knowing multiple checksums does not help the attacker in generating corrupted packets that will pass the checksum verification for future checksums.

Additional Remarks. Attackers may try to attack the DART scheme itself by preventing nodes from receiving checksum packets. Recall that the checksum packets are flooded among all forwarder nodes. If attackers are always able to prevent a node from receiving checksum packets, this implies that the node is completely surrounded by attackers. In this case, the attackers can isolate the node by dropping all data packets, thus achieving the same effect as dropping checksums. Our DART scheme can provide additional resiliency against checksum dropping by flooding the checksum not only among the set of forwarder nodes, but also among the nodes that are near forwarder nodes (*e.g.*, within two hops).

Size of the security parameters. As shown in Theorems 1 and 2, we can reduce the false negative probability of the verification by using a large field size q or a large security parameter b. However, using a large field size also results in large symbol sizes, causing larger network overhead (since the ratio n/m increases for a fixed packet size). Security parameter b allows us to increase the security of the scheme without increasing the field size.

For a typical field size of $q = 2^8$, selecting $b = 2$ is sufficient. With individual packet verification, if an attacker injects more than $256^2 = 65,536$ packets, then on average, only one polluted packet will be forwarded more than one hop away. Our experiments confirm that selecting $b = 2$ is sufficient to contain pollution attacks.

6. THE EDART SCHEME

In our DART scheme, valid packets received by a node are unnecessarily delayed until the next checksum arrives. Ideally, nodes should delay only polluted packets for verification, whereas unpolluted packets should be mixed and forwarded without delay. However, nodes do not know which packets are polluted before receiving a checksum packet and are faced with a dilemma: Imprudent forwarding may pollute a large portion of the network, while overstrict verification will unnecessarily delay valid packets.

We propose EDART, an adaptive verification scheme which allows nodes to optimistically forward packets without verifying them. As in DART, nodes verify packets using the periodic checksums. But in EDART, only nodes near the attacker tend to delay packets for verification, while nodes farther away tend to forward packets without delaying. Therefore, pollution is contained to a limited region around the attacker and correct packets are forwarded without delay in regions without attackers. A major advantage of EDART is that, when no attacks exist in the network, the packets are delivered without delay, incurring almost no impact on the system performance. Below, we describe EDART and provide bounds on the attack impact, the attack success frequency, and the packet delivery delay in the network.

6.1 Scheme Description

In EDART, each node is in one of two modes, *forward mode* or *verify mode*. In verify mode, a node delays received packets until they can be verified using the next checksum. In forward mode, a node mixes and forwards received packets immediately without verification, except if the packet has traveled more than a predetermined number of hops since its last verification. The limited scope of any unverified packets ensures that the maximum number of hops a polluted packet can travel is bounded. As in DART, upon receipt of a checksum, nodes always verify any buffered unverified packet whose receive time is before the checksum creation time.

Nodes start in the forward mode at system initialization. A node switches to the verify mode upon detecting a verification failure. The amount of time a node stays in the verify mode is a decreasing function of its distance to an attacker node, so that nodes near an attacker tend to verify packets, while nodes farther away tend to forward packets without delay.

The detailed pseudo-code for EDART is presented in Algorithm 1. Each network coded packet contains a new field, h_v, which records the number of hops the packet has traveled since its last verification. Each node maintains a variable C_v (the verification counter), indicating the amount of time that the node will stay in the verify mode (*e.g.*, $C_v = 0$ means that the node is in the forward mode). A node also maintains two sets of packets, forward_set and delay_set. Packets in forward_set can be combined to form coded packets, while packets in delay_set are held for verification.

At system initialization, each node starts in the forward mode (*i.e.*, $C_v = 0$) and both forward_set and delay_set are empty.

Upon receiving a coded packet, a node adds the packet to the delay_set if the node is in the verify mode (*i.e.*, $C_v > 0$) or if the packet has traveled more than δ hops since its last verification (*i.e.*, if $h_v > \delta$, where δ is a pre-determined system parameter). Otherwise, the packet is added to the forward_set for immediate forwarding. A new coded packet is formed by combining packets in the forward_set. The h_v field of the new packet is set to $h_{\max} + 1$, where $h_{\max}$ is the maximum h_v among all packets that are combined to form this new packet.

Upon receiving a checksum packet, a node verifies all unverified packets in both forward_set and delay_set. If all packets pass the verification, it decrements C_v by one (unless it is already 0). If there are packets that fail the verification, the node increments C_v by $\alpha(1-\frac{h_{\min}}{\delta})$, where $h_{\min}$ is the minimum h_v of all the packets that fail the verification and α is a pre-determined system parameter. For all packets that pass the verification, their h_v field is reset to 0.

Note that the h_v field does not require integrity protection. On the one hand, if the attacker sets h_v large, then the polluted packets are only propagated over a small number of hops. On the other hand, if the attacker sets h_v small, then the neighbors of the attacker will stay in the verify mode longer after checksum verification, preventing pollution from the attacker node for a longer duration of time. In the next section, we show that regardless of how attackers may set the h_v value, the overall attack impact is still bounded.

6.2 Security Analysis

We now analyze the properties of EDART, first in the context of one attacker, and then extend the analysis to the case of multiple attackers. We refer to the time between two consecutive checksum creation events as a *time interval*. To measure the severity of a pollution attack, we define the following metrics:

– *pollution scope*: The number of hops traveled by a polluted packet before being discarded, which captures the maximum impact caused by a single polluted packet. We also consider the *average pollution scope* which captures the impact of an attack averaged over time.

– *pollution success frequency*: The frequency of pollution attacks with scope greater than one. Note that an attack with pollution scope of one hop has no impact on the network, as the polluted packet is dropped immediately by the first hop neighbors of the attacker.

To measure the effectiveness of EDART in minimizing packet delivery delay we define *unnecessary delay* as the number of additional time intervals a node stays in the verify mode, compared to an ideal scheme where only the direct neighbors of an active attacker are in the verify mode and all other nodes are in the forward mode.

In EDART, attackers can only increase the pollution scope of an attack at the cost of decreasing their pollution success frequency

Algorithm 1 Adaptive Verification Scheme

Executed at system initialization

1: $C_v = 0$; forward_set $= \emptyset$; delay_set $= \emptyset$

Executed on receiving packet p

1: **if** ($C_v > 0$ or $h_v \geq \delta$) **then** add p to delay_set
2: **else** add p to forward_set

Executed to output a packet

1: Select a subset of packets from forward_set to form a coded packet as required by the particular network coding system.
2: Set h_{max} to be maximum h_v in the selected packets
3: For the coded packet, set $h_v = h_{\text{max}} + 1$

Executed on receiving checksum (CHK, s, t)

1: Verify all unverified safe packets in both forward_set and delay_set against CHK
2: Set $h_v = 0$ for all verified packets
3: **if** there exist invalid packets that failed verification **then**
4: Set h_{min} as the minimum h_v in all packets that failed verification
5: $C_v = C_v + \alpha(1 - \frac{h_{\text{min}}}{\delta})$
6: **else if** $C_v > 0$ **then**
7: $C_v = C_v - 1$

and vice-versa: Setting h_v to a low value for a polluted packet will result in a larger pollution scope, but the direct neighbors will isolate the attacker for a longer period. Hence, the overall severity of the attack is bounded. We now present properties that precisely capture the effectiveness of the EDART scheme for the case of one attacker.

PROPERTY 1. *The maximum pollution scope of an attack is upper-bounded by $\delta + 1$.*

PROOF. Each honest node increments by one the h_v field of its newly coded packets. Then, clearly, a polluted packet that was forwarded by δ honest nodes will have $h_v \geq \delta$; thus, it will be verified, detected and dropped by the next honest node. □

PROPERTY 2. *The average pollution scope per time interval is upper-bounded by δ/α.*

PROOF. Let h be the minimum h_v value of polluted packets sent by an attacker in the current time interval. Then, the maximum pollution scope of polluted packets in this time interval is $\delta - h$. Upon receiving the first checksum, all the direct neighbors of the attacker will detect verification failures, and increment their C_v by $\alpha(1 - \frac{h}{\delta})$. Thus, they will stay in the verify mode for at least $\alpha(1 - \frac{h}{\delta})$ time intervals. As long as all the neighbors of the attacker are in the verify mode, all the polluted packets generated by the attacker will be detected and dropped at the first hop, thus causing no pollution effect on the network. Therefore, the average pollution scope per time interval is at most $(\delta - h)/(\alpha(1 - \frac{h}{\delta}) + 1) < (\delta - h)/(\alpha(1 - \frac{h}{\delta})) = \delta/\alpha$. □

PROPERTY 3. *The maximum pollution success frequency is upper-bounded by δ/α.*

PROOF. When an attacker sends polluted packets with $h_v = h$ in some time interval, for the next $\alpha(1 - \frac{h}{\delta})$ time intervals its pollution attacks will be ineffective (polluted packets will be verified and dropped by its first-hop neighbors). Thus, its pollution success frequency is at most $1/(\alpha(1 - \frac{h}{\delta}) + 1)$. To maximize this value, it sets $h = \delta - 1$, resulting in the maximum success frequency of $1/(\alpha(1 - \frac{\delta-1}{\delta}) + 1) < \delta/\alpha$. □

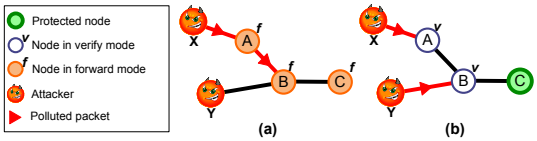


Figure 2: (a) Attacker X attacks first and nodes A and B receive polluted packets; (b) In the following time interval, A and B switch to verify mode, and attacker Y starts attacking. Y's polluted packets are immediately dropped by node B and further nodes (node C) are protected. Thus, the strength of Y's attack is diminished by X's attack.

PROPERTY 4. *Let h be the minimum h_v value of polluted packets sent by an attacker. Nodes at i hops away from the attacker (for $2 \leq i \leq \delta - h - 1$) have unnecessary delay of $\alpha(1 - \frac{h+i}{\delta})$ time intervals. Nodes more than $\delta - h - 1$ hops away do not have unnecessary delay.*

PROOF. Since the h_v value is incremented at each hop, nodes that are i hops away from the attacker (with $2 \leq i \leq \delta - h - 1$) stay in the verify mode for $\alpha(1 - \frac{h+i}{\delta})$ time intervals. Nodes that are more than $\delta - h - 1$ hops away do not switch to the verify mode since polluted packets are verified and dropped by nodes $\delta - h$ away from attacker. □

Multi-attacker case: With multiple attackers, the attack strength per attacker in terms of maximum pollution scope, average pollution scope, and maximum success frequency is still bounded as in Properties 1, 2, and 3. To see this, we examine two different cases. First, we consider attackers that are far apart from each other (*e.g.*, over 2δ hops away) such that a honest node is only in the pollution scope of one attacker. In this case, we can view the network subdivided into smaller areas, and in each smaller area there is only one attacker, thus the bounds in Properties 1, 2, and 3 still hold. Second, we consider attackers positioned such that some honest nodes are affected by multiple attackers. The reaction of such honest nodes is driven by their closest attacker. As shown in the example of Fig. 2, the effectiveness per attacker is reduced, because nearby attackers cancel the effects of each other. Thus, Properties 1, 2, and 3 also hold in this case. Our experiments in Sec. 7 confirm that EDART remains effective against pollution attacks in the presence of multiple attackers.

6.3 Selection of δ and α

δ is defined as the number of hops after which a coded packet is always verified. α is a parameter that controls the amount of time a node stays in the verify mode. By Properties 1, 2, and 3, the scope and success frequency of an attack are directly proportional to δ and inversely proportional to α, thus we can increase attack resiliency by selecting a small δ and a large α. However, a small δ and a large α result in a larger packet delay: A small δ causes valid packets to travel only a small number of hops before being delayed for verification, and a large α causes a larger unnecessary delay in the presence of attacks (Property 4). Thus, we need to balance between attack resiliency and packet delivery delay when selecting δ and α.

For systems that can tolerate large delivery latency, such as large file transfers in mesh networks or code updates in sensor networks, we can use a small δ and a large α to increase the system resiliency. On the other hand, for systems that are sensitive to delivery latency, such as video or audio streaming, we can use a large δ and a small α to reduce delay. We also note that in a benign network, the value of δ determines the delivery latency. Thus, in a network in which attacks are rare, we can use a large δ to reduce delivery latency in normal cases, and use a large α to limit the attack impact when under attack.

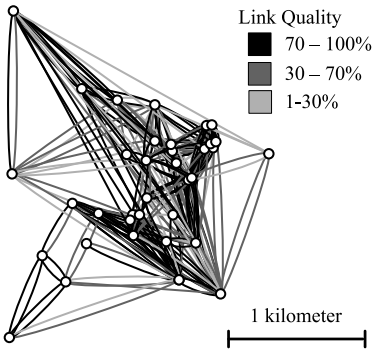


Figure 3: Roofnet topology and link qualities

7. EXPERIMENTAL EVALUATION

We show through simulations the impact of pollution attacks on an unprotected system and the high cost of current cryptographic-based solutions. We then perform an evaluation of our defense schemes. Our experiments are based on the well-known MORE protocol [3], a network coding-based routing protocol for wireless mesh networks. We selected MORE because it is one of the best known network coding based routing protocols for wireless networks and the source code is publicly available. We use the real-world link quality measurements from Roofnet [38], an experimental 802.11b/g mesh network in development at MIT. Roofnet has also been widely used in other research papers [22, 52–55] as a representative wireless mesh network testbed.

7.1 MORE in a Nutshell

MORE is a routing protocol for wireless mesh networks that uses network coding to achieve increased performance. MORE sends data in generations of n packets. For each generation, the source continuously broadcasts coded packets of native packets from the current generation until it receives an acknowledgment from the destination; the source then moves on to the next generation. The source includes in the header of each coded packet the *forwarder set*, which is a set of nodes that participate in forwarding packets; this set is obtained based on each node's distance to the destination.

A forwarder node stores an overheard packet only if the packet is linearly independent with previously stored packets. The reception of a new coded packet also triggers the node to broadcast ℓ new coded packets, where ℓ is determined based on the node's relative distance to the destination compared to other nodes. The new coded packets are generated from the stored coded packets previously received for the same generation.

When the destination receives n linearly independent coded packets, it can decode the generation and recover the native packets, upon which it sends an acknowledgment to the source to indicate that it can start sending the next generation.

7.2 Experimental Methodology

Simulation Setup. Our experiments are performed with the Glomosim simulator [56] configured with 802.11 as the MAC layer protocol. We use realistic link quality measurements for the physical layer of Glomosim to determine the link loss rate. Specifically, we use the real-world link quality trace data from the Roofnet testbed, publicly available at [38]. The network consists of 38 nodes, and the topology and link qualities are shown in Fig. 3. The raw bandwidth is 5.5Mbps.

We assume the clocks of nodes in the network are loosely synchronized, with the maximum clock drift between any node and the source being $\Delta = 100$ms[3]. We use RSA [58] digital signatures with 1024-bit keys and simulate delays to approximate the performance of a 1.3 GHz Intel Centrino processor.

We use the MORE unicast protocol to demonstrate our schemes. In each experiment, we randomly select a pair of nodes as the source and destination. The source starts to transfer a large file to the destination 100 seconds after the experiment starts for a duration of 400 seconds. The CDF results shown in our graphs are taken over 200 randomly selected source-destination pairs.

MORE Setup. We use the default MORE setup as in [3]: The finite field for network coding is $\mathbb{F}_{2^8}$, the generation size n is 32 packets, and the packet size is 1500B.

Attack Scenario. We vary the number of attackers from 1 to 10 (out of a total of 38 nodes). Since in MORE only forwarder nodes and nodes in the range of a forwarder node can cause packet pollution, we select attackers at random among these nodes. If the total number of such nodes is less than the specified number of attackers, we select all of them as attackers.

The attackers inject polluted packets, but follow the protocol otherwise. To examine the impact of the attack, we define *pollution intensity* (PI) as the average number of polluted packets injected by the attacker for each packet it receives. Thus, it captures the frequency of pollution attacks from an attacker. We vary PI to examine the impact of different levels of attack intensity.

DART and EDART Setup. We set the checksum parameter $b = 2$, which results in a checksum size of 64 bytes. The source broadcasts a checksum packet after broadcasting every 32 data packets[4]. We use the pipelining technique described in Sec. 5.3, with the pipeline size of 5. For EDART, we use $\delta = 8$ and $\alpha = 20$. These parameters are experimentally selected to suit the small scale of the network and to balance between overhead, throughput, and latency.

As a baseline, we use a hypothetical ideal defense scheme referred to as *Ideal*, where the polluted packets are detected with zero cost and immediately dropped by a node. Note that under benign conditions, the *Ideal* scheme behaves the same as the original MORE protocol. We compare our schemes with *Ideal* using the same pipeline size to examine the latency caused by delayed packet verification.

Metrics. In our experiments, we measure throughput, latency and overhead (bandwidth and computation). *Throughput* is measured as the average receiving rate at which the destination receives data packets (after decoding). *Latency* is measured as the delay in receiving the first packet (decoded) at the destination. The only bandwidth overhead incurred by our scheme is the dissemination of checksum packets. We measure the *bandwidth overhead* as the average bandwidth overhead per node among all forwarder nodes that forward checksum packets. Since intermediate nodes perform only digital signature and checksum verification, both of which incur small overhead (checksum verification overhead is demonstrated in our micro-benchmark below), we measure the *computational overhead* as the number of digital signatures per second performed by the source for signing checksum packets. The overhead measurement does not include the overhead due to time synchronization.

7.3 Impact of Pollution Attacks

To demonstrate the severity of pollution attacks on network coding, we evaluate the impact of the attack conducted by a **single attacker**. Fig. 4 shows the throughput of MORE in a network with only one attacker with various pollution intensities. In the no at-

[3]Current secure clock synchronization schemes [50, 57] can achieve clock drift in the order of microseconds. We use a much larger clock drift to demonstrate that our schemes only require loose clock synchronization.

[4]This does *not* mean there is only one checksum per generation. In MORE, the source keeps broadcasting coded packets for a generation (usually more than 32 packets) until the destination is able to decode the entire generation of packets.

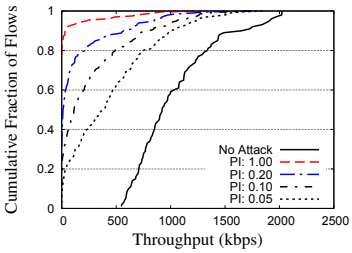


Figure 4: The throughput CDF in the presence of a single attacker for various pollution intensities (PIs). Even when the attacker injects only one polluted packet every 20 received packets (PI=0.05%), the impact of the attack is still very significant.

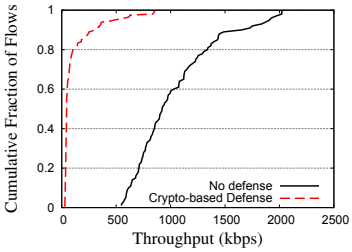


Figure 5: Impracticality of previous work: Throughput CDF of original MORE and of MORE with cryptographic-based defense in a benign network. Even when no attack takes place, previous schemes have a very high overhead that makes them impractical.

tack case, we observe that all the flows have a throughput greater than 500kbps, with median at around 1000kbps. In the attack case with pollution intensity of 1, the throughput of around 80% of all flows goes to zero, and 97% of all flows have throughput less than 500kbps. Even when the pollution intensity is very small at 0.05 (*i.e.*, the attacker only injects, on average, one polluted packet per 20 packets received), the throughput of most flows still degrades significantly, with around 60% of all flows having a throughput below 500kbps. Therefore, we conclude that pollution attacks are extremely detrimental to the system performance, even when performed very infrequently and by only one attacker.

7.4 Limitations of Previous Solutions

We use a benign scenario to show that previous cryptographic-based solutions are not practical for wireless networks. Protocols that add a significant overhead to the system, even when **no attack occurs**, provide little incentive to be used.

We set up our experiments to strongly favor the cryptographic-based solutions as follows. We only account for computational overhead at the intermediate nodes, and ignore all other overhead, such as the computational overhead at the source and the bandwidth overhead required to disseminate digital signatures and/or homomorphic hashes. We also use a large symbol size of 20 bytes (hence $m = 1500/20 = 75$) to favor these schemes in reducing their computational overhead. Any practical network coding system requires $n \ll m$ so that the network overhead of network coding is small. With the generation size $n = 32$ and $m = 75$, the relative network overhead is already around $\rho = 42\%$. We also discount such large overhead of network coding for these schemes. Finally, we use batch verification, such that each node can batch verify a set of packets at the cost of one verification, and use the pipelining technique (with pipeline size of 5) described in Sec. 5.3 to further boost the performance of such schemes.

Fig. 5 shows the throughput CDF of strongly-favored cryptographic-based schemes and the original MORE protocol in a network with no attackers. We see that even when being exceedingly favored, the large computational overhead of these schemes still results in significant throughput degradation, with 80% of the flows have throughput 100kbps or less[5] and the median throughput degrades by 96%. Hence, we conclude they are impractical for wireless mesh networks.

Size parameter (b value)	1	2	3
Generation time (ms)	0.475	0.957	1.432
Per packet verification time (ms)	0.188	0.388	0.507
Batch verification time (ms)	0.492	1.319	2.458

Table 1: Computational cost for checksum generation and verification for different checksum sizes. Batch verification time is for verifying 32 packets.

7.5 Evaluation of DART and EDART

We present an evaluation of the performance of our proposed defenses, DART and EDART. We first perform micro-benchmarks to evaluate the computational cost of checksum generation and verification. We then evaluate the performance of our defense schemes for benign networks and for networks under various pollution attack intensities. Finally, we examine their bandwidth and computational overhead.

Micro-benchmarks. We evaluate the computational cost of checksum generation and verification on a computer with 1.3 GHz Intel Centrino processor, and use the random number generator in the OpenSSL library (version 0.9.8e) for generating the random checksum coefficients. Table 1 summarizes the results for generating and verifying checksums of different sizes.

Benign networks. Fig. 6 shows the throughput and latency of our schemes in a benign network with no attackers, as compared to the MORE protocol. In Fig. 6(a), we see that DART incurs some throughput degradation (around 9% degradation when comparing median throughputs), whereas EDART incurs almost no degradation.

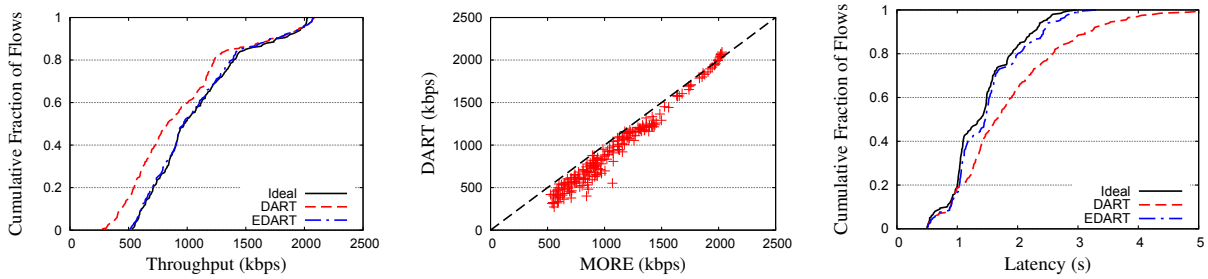
Fig. 6(b) provides insights into the throughput degradation of DART by showing the scatter plot of throughput for DART with respect to the MORE protocol. We see that the throughput degradation is more severe for flows with smaller throughput, while flows with higher throughput are less affected by DART. This is because the throughput degradation of DART is primarily caused by the checksum authentication delay at intermediate nodes. Flows with smaller throughput typically have a longer path length, hence they incur a larger aggregate authentication delay and consequently higher throughput degradation.

In Fig. 6(c), we observe a similar pattern for the latency of DART and EDART. DART incurs an additional 0.4 second in median latencies compared to the *Ideal* scheme with the same pipeline size, while EDART incurs almost no additional latency. For similar reasons to the throughput, we also observe that for DART, the latency overhead is larger for flows that already have a large latency.

In summary, when no attacks take place, both of our schemes have throughput over 20 times higher than cryptographic-based schemes and cause minimal degradation on system performance. The performance of EDART is almost identical to the *Ideal* scheme.

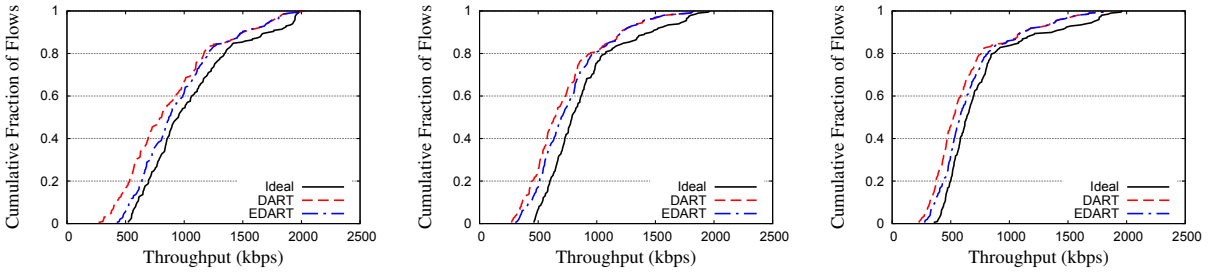
Networks under attack. We examine the effectiveness of our defense against different number of pollution attackers. Fig. 7 shows the throughput CDF for the case of 1, 5, and 10 attackers with pollution intensity of 0.2. We see that across different number of attackers, the throughput only degrades slightly compared to the *Ideal* scheme. EDART improves the throughput over DART, especially

[5] Some flows have throughput greater than the maximum throughput shown in Sec. 5 because we discounted the network coding overhead in the results.



(a) Throughput CDF (b) Throughput scatter plot (c) Latency CDF

Figure 6: The throughput and latency of DART and EDART under benign case.



(a) 1 attacker (b) 5 attackers (c) 10 attackers

Figure 7: Throughput CDF of our defense schemes in the presence of pollution attackers

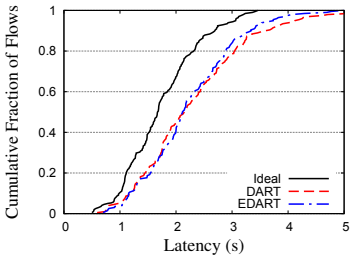
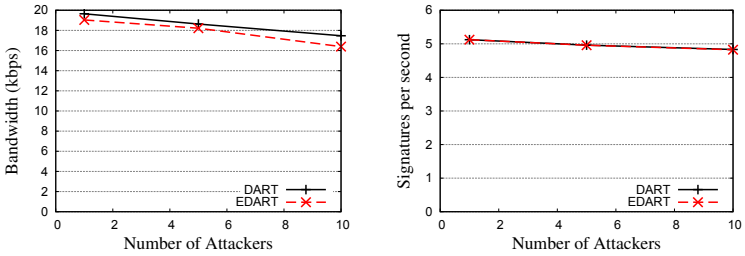


Figure 8: Latency CDF of DART and EDART in the presence of 5 pollution attackers



(a) Bandwidth overhead (b) Computational overhead at the source

Figure 9: Bandwidth and computational overhead of DART and EDART

for low throughput flows in the one attacker case. In Fig. 7(a), the throughput improvement of EDART over DART is most significant for flows with throughput below the median, which typically traverse many forwarder nodes. In EDART a single attacker only influences a small portion of the forwarder nodes to delay packets, accounting for the greater throughput improvement.

Fig. 8 shows the latency of DART and EDART in the case of 5 attackers with the pollution intensity of 0.2. The figures for the 1 and 10 attackers scenarios are similar and were omitted. We see that the median latency increase of DART and EDART over the *Ideal* scheme is around 0.5 seconds. This confirms that the overall latency due to checksum verification is small.

An apparent anomaly is that EDART does not have a much smaller latency than DART, although in EDART nodes forward packets optimistically without delay. This is because the latency metric accounts for the delay of the first generation. In EDART, the first generation of packets will be delayed as in DART because, although nodes start in the forward mode, the propagation of the initial polluted packets causes all forwarder nodes to switch to the verify mode. However, for all later generations, only neighbors of the attackers delay packets in EDART. Thus, the delay of later generations is smaller, leading to the improved throughput of EDART over DART.

We also examined the throughput and latency for other pollution intensities, all of which show similar results as with the pollution intensity of 0.2. For larger pollution intensities, the congestion effect of polluted packets also causes a certain level of throughput degradation; however, our defense mechanisms still maintain a level of performance similar to the *Ideal* scheme.

Overhead. Figs. 9(a) and 9(b) show the bandwidth and computational overhead of our defense schemes, respectively. Both the bandwidth and computational overhead remain at a stable level across different number of attackers, 18kbps per forwarder node and 5 signatures per second at the source, respectively. This is because our checksum generation and dissemination is independent of the number of attackers. The bandwidth overhead of 18kbps is less than 2% of the throughput achieved by the system on average.

It may also be counter-intuitive that both the bandwidth and computational overhead decreases slightly when the number of attackers increases. The reason is that the frequency that the source disseminates packets decreases slightly when there are more attackers, due to the congestion effect of the polluted packets. This results in slightly fewer checksums being generated and disseminated.

8. CONCLUSION

In this paper, we present two new and practical defense schemes, DART and EDART, against pollution attacks in intra-flow network coding systems for wireless mesh networks. DART combines random linear transformations with time asymmetry in checksum verification to efficiently prevent packet pollution. EDART incorporates optimistic packet forwarding to reduce delivery latency and improve system performance. Besides providing a detailed security analysis and analytical bounds for our schemes, we demonstrate their practicality through simulations that use a well-known

network coding routing protocol for wireless mesh networks and real-life link quality measurements from a representative testbed for mesh networks. Our results demonstrate that:

- The effect of pollution attacks is devastating in mesh networks using intra-flow network coding. Without any protection, a single attacker can reduce the throughput of 80% of all flows to zero.

- Previous solutions are impractical for wireless mesh networks. Even when no attackers are present in the system, the large overhead of previous cryptographic-based schemes result in as much as 96% degradation in the system throughput.

- Our schemes provide an efficient defense against pollution attacks and incur only a small computational and bandwidth overhead. When no attackers are present, DART incurs a small performance degradation, while EDART achieves almost the same performance as the original MORE protocol. When the system is under attack, both schemes lead to a performance level similar to a hypothetical *Ideal* defense scheme.

9. REFERENCES

[1] R. Ahlswede, N. Cai, S.-Y. Li, and R. Yeung, "Network information flow," *Information Theory, IEEE Transactions on*, vol. 46, no. 4, pp. 1204–1216, 2000.

[2] S. Katti, D. Kabati, W. Hu, H. Rahul, and M. Medard, "The importance of being opportunistic: Practical network coding for wireless environments," in *In Proc. of Allerton Conf. on Commun. Control and Computing*, Oct. 2005.

[3] S. Chachulski, M. Jennings, S. Katti, and D. Katabi, "Trading structure for randomness in wireless opportunistic routing," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 169–180, 2007.

[4] C. Gkantsidis and P. Rodriguez, "Network coding for large scale content distribution," in *In Proc. IEEE Infocom*, Mar. 2005.

[5] A. G. Dimakis, P. B. Godfrey, M. J. Wainwright, and K. Ramchandran, "The benefits of network coding for peer-to-peer storage systems," in *Third Workshop on Network Coding, Theory, and Applications*, 2007.

[6] C.Fragouli and A.Markopoulou, "A network coding approach to overlay network monitoring," in *Allerton 2005*.

[7] C. Fragouli and A. Markopoulou, "Network coding techniques for network monitoring: a brief introduction," in *Intl Zurich Seminar on Commun.*, 2006.

[8] T. Ho, B. Leong, Y.-H. Chang, Y. Wen, and R. Koetter, "Network monitoring in multicast networks using network coding," in *ISIT*, 2005.

[9] M. Effros, T. Ho, and S. Kim, "A tiling approach to network code design for wireless networks," in *IEEE Information Theory Workshop*, 2006.

[10] J. Jin, T. Ho, and H. Viswanathan, "Comparison of network coding and non-network coding schemes for multi-hop wireless networks," in *ISIT 2006*.

[11] A. F. Dana, R. Gowaikar, R. Palanki, B. Hassibi, and M. Effros, "Capacity of wireless erasure networks," *IEEE Trans. on Information Theory*, vol. 52, 2006.

[12] S. Deb and M. Medard, "Algebraic gossip: A network coding approach to optimal multiple rumor mongering," *IEEE Trans. on Info. Theory*, 2006.

[13] J. Widmer and J.-Y. L. Boudec, "Network coding for efficient communication in extreme networks," in *WDTN 2005*.

[14] D. S. Lun, M. Médard, R. Koetter, and M. Effros, "Further results on coding for reliable communication over packet networks," in *ISIT*, 2005.

[15] Y. W. P. A. Chou and S.-Y. Kung, "Minimum-energy multicast in mobile ad hoc networks using network coding," *IEEE Transactions on Communications*, 2005.

[16] D. S. Lun, N. Ratnakar, R. Koetter, M. M. edard, E. Ahmed, and H. Lee, "Achieving minimum cost multicast: A decentralized approach based on network coding," in *Proceeding of IEEE Infocom*, 2005.

[17] J. Widmer, C. Fragouli, and J.-Y. L. Boudec, "Energy-efficient broadcasting in wireless ad-hoc networks," in *Netcod 2005, Italy*, April 2005.

[18] K. Jain, "On the power (saving) of network coding," in *Allerton*, 2005.

[19] T. Ho, "On constructive network coding for multiple unicasts," in *44th annual Allerton Conference on Communication, Control and Computing*, 2006.

[20] D. Traskov, N. Ratnakar, D. S. Lun, R. Koetter, and M. Médard, "Network coding for multiple unicasts: An approach based on linear optimization," in *Proceedings of the International Symposium on Information Theory*, 2006.

[21] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft, "Xors in the air: practical wireless network coding," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 4, pp. 243–254, 2006.

[22] B. Radunovic, C. Gkantsidis, S. G. P. Key, W. Hu, and P. Rodriguez, "Multipath code casting for wireless mesh networks," Microsoft Research, Technical Report MSR-TR-2007-68, March 2007.

[23] J.-S. Park, M. Gerla, D. S. Lun, Y. Yi, and M. Medard, "Codecast: a network-coding-based ad hoc multicast protocol," *IEEE Wireless Comm.*, 2006.

[24] M. Médard, M. Effros, T. Ho, and D. R. Karger, "On coding for non-multicast networks," in *Allerton*, 2003.

[25] I.-H. Hou, Y.-E. Tsai, T. Abdelzaher, and I. Gupta, "Adapcode: Adaptive network coding for code updates in wireless sensor networks," in *INFOCOM*, 2008.

[26] L. Li, R. Ramjee, M. Buddhikot, and S. Miller, "Network coding-based broadcast in mobile ad-hoc networks," *Proc. of INFOCOM 2007*.

[27] C. Fragouli, J. Widmer, and J.-Y. Le Boudec, "A network coding approach to energy efficient broadcasting: From theory to practice," *INFOCOM 2006*.

[28] J. Dong, R. Curtmola, R. Sethi, and C. Nita-Rotaru, "Toward secure network coding in wireless networks: Threats and challenges," in *NPSec*, 2008.

[29] D. Charles, K. Jain, and K. Lauter, "Signatures for network coding," *40th Annual Conference on Information Sciences and Systems*, 2006.

[30] Z. Yu, Y. Wei, B. Ramkumar, and Y. Guan, "An efficient signature-based scheme for securing network coding against pollution attacks," in *Proceedings of INFOCOM 08*, Phoenix, AZ, April 2008.

[31] F. Zhao, T. Kalker, M. Medard, and K. Han, "Signatures for content distribution with network coding," *ISIT 2007*.

[32] Q. Li, D.-M. Chiu, and J. Lui, "On the practical and security issues of batch content distribution via network coding," *Proc. of ICNP '06*, Nov. 2006.

[33] M. Krohn, M. Freedman, and D. Mazieres, "On-the-fly verification of rateless erasure codes for efficient content distribution," *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*, pp. 226–240, 9-12 May 2004.

[34] C. Gkantsidis and P. Rodriguez Rodriguez, "Cooperative security for network coding file distribution," *Proc. of INFOCOM 2006*.

[35] T. Ho, B. Leong, R. Koetter, M. Medard, M. Effros, and D. Karger, "Byzantine modification detection in multicast networks using randomized network coding," *ISIT 2004*.

[36] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, and M. Medard, "Resilient network coding in the presence of byzantine adversaries," *INFOCOM 2007*.

[37] D. Wang, D. Silva, and F. R. Kschischang, "Constricting the adversary: A broadcast transformation for network coding," *Allerton 2007*, 2007.

[38] "MIT roofnet." http://pdos.csail.mit.edu/roofnet/doku.php.

[39] D. Boneh, D. Freeman, J. Katz, and B. Waters, "Signing a linear subspace: Signature schemes for network coding," in *Proc. of PKC '09*, 2009.

[40] D. Silva, F. Kschischang, and R. Koetter, "A rank-metric approach to error control in random network coding," *IEEE Inf. Theory for Wireless Ntwks*, 2007.

[41] R. Koetter and F. R. Kschischang, "Coding for errors and erasures in random network coding," *Information Theory, IEEE Transactions on*, 2008.

[42] R. W. Yeung and N. Cai, "Network error correction, part i: basic concepts and upper bounds," *Commun. Inf. Syst.*, vol. 6, no. 1, pp. 19–36, 2006.

[43] N. Cai and R. W. Yeung, "Network error correction, part ii: lower bounds," *Commun. Inf. Syst.*, vol. 6, no. 1, pp. 37–54, 2006.

[44] P. Chou and Y. Wu, "Network coding for the internet and wireless networks," *Signal Processing Magazine, IEEE*, vol. 24, no. 5, pp. 77–85, Sept. 2007.

[45] Y. Lin, B. Li, and B. Liang, "Efficient network coded data transmissions in disruption tolerant networks," in *Proc. of INFOCOM 2008*.

[46] T. Cui, L. Chen, and T. Ho, "Energy efficient opportunistic network coding for wireless networks," in *Proceedings of INFOCOM 08*, Phoenix, AZ, April 2008.

[47] A. Perrig, R. Canetti, J. D. Tygar, and D. Song, "The TESLA broadcast authentication protocol," *RSA CryptoBytes*, vol. 5, no. Summer, 2002.

[48] A. Perrig, R. Canetti, D. Song, and D. Tygar, "Efficient and secure source authentication for multicast," in *Proc. of NDSS '01*, 2001.

[49] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler, "Spins: security protocols for sensor networks," *Wireless Networks*, vol. 8, no. 5, 2002.

[50] K. Sun, P. Ning, and C. Wang, "Secure and resilient clock synchronization in wireless sensor networks," *JSAC*, vol. 24, no. 2, Feb. 2006.

[51] J. Dong, R. Curtmola, and C. Nita-Rotaru, "Practical defenses against pollution attacks in intra-flow network coding for wireless mesh networks," Purdue University, Technical Report, 2009.

[52] D. S. J. D. Couto, D. Aguayo, J. Bicket, and R. Morris, "A high-throughput path metric for multi-hop wireless routing," in *Proc. of ACM MobiCom 2003*.

[53] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris, "Link-level measurements from an 802.11b mesh network," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, pp. 121–132, 2004.

[54] J. Bicket, D. Aguayo, S. Biswas, and R. Morris, "Architecture and evaluation of an unplanned 802.11b mesh network," in *Proc. of ACM MobiCom 2005*.

[55] S. Biswas and R. Morris, "Opportunistic routing in multi-hop wireless networks," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 1, pp. 69–74, 2004.

[56] "Glomosim," http://pcl.cs.ucla.edu/projects/glomosim/.

[57] K. Sun, P. Ning, and C. Wang, "Tinysersync: secure and resilient time synchronization in wireless sensor networks," in *Proc. of ACM CCS 2006*.

[58] *Digital Signature Standard (DSS)*. National Institute for Standards and Technology (NIST), 2006, no. FIPS 186-3.