Practical Defenses Against Pollution Attacks in Wireless Network Coding

JING DONG, Purdue University
REZA CURTMOLA, New Jersey Institute of Technology
CRISTINA NITA-ROTARU, Purdue University

Recent studies have shown 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 article, we address pollution attacks against network coding systems in wireless mesh networks. We demonstrate that previous solutions are impractical in wireless networks, incurring an unacceptable 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. We also propose efficient attacker identification schemes for both DART and EDART that enable quick attacker isolation and the selection of attacker-free paths, achieving additional performance improvement. A detailed security analysis shows that the probability of a polluted packet passing our verification procedure is very low (less than 0.002% in typical settings). 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 with previous solutions.

Categories and Subject Descriptors: C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communication*; C.2.0 [**Computer-Communication Networks**]: General—*Security and protection (e.g., firewalls)*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms: Performance, Security

Additional Key Words and Phrases: Network coding, pollution attacks, network coding security, wireless network security, security

ACM Reference Format:
Dong, J., Curtmola, R., and Nita-Rotaru, C. 2011. Practical defenses against pollution attacks in wireless network coding. ACM Trans. Info. Syst. Sec. 14, 1, Article 7 (May 2011), 31 pages.
DOI = 10.1145/1952982.1952989 http://doi.acm.org/10.1145/1952982.1952989

This research is sponsored in part by the U.S. National Science Foundation CAREER grant 0545949-CNS.

Authors' addresses: J. Dong, Knight Equity Markets, 575 Market Street, San Francisco, CA 94105; email: jingdong@gmail.com; R. Curtmola, Department Computer Science, New Jersey Institute of Technology, GITC4301, 218 Central Ave., Newark, NJ 07102; email: crix@njit.edu; C. Nita-Rotaru, Computer Science Department, Purdue University, LWSN 2142J, 305 N. University St., West Lafayette, IN 47907; email: crisn@cs.purdue.edu.

DOI 10.1145/1952982.1952989 http://doi.acm.org/10.1145/1952982.1952989

1. INTRODUCTION

Network coding [Ahlswede et al. 2000] introduces a new paradigm for network protocols. Recent research has demonstrated the advantages of network coding through practical systems such as COPE [Katti et al. 2005] and MORE [Chachulski et al. 2007] for wireless unicast and multicast, Avalanche [Gkantsidis and Rodriguez 2005] for P2P content distribution, and protocols for P2P storage [Dimakis et al. 2007] and for network monitoring and management [Fragouli and Markopoulou 2005, 2006; Ho et al. 2005]. Network coding has been shown to increase throughput [Effros et al. 2006; Jin et al. 2006; Dana et al. 2006], reduce network congestion [Deb and Medard 2006], increase reliability [Widmer and Boudec 2005; Lun et al. 2005a], and reduce power consumption [Chou and Kung 2005; Lun et al. 2005b; Widmer et al. 2005; Jain 2005], in unicast [Ho 2006; Traskov et al. 2006; Chachulski et al. 2007; Katti et al. 2006; Radunovic et al. 2007], multicast [Park et al. 2006; Chachulski et al. 2007], and more general network configurations [Médard et al. 2003; Hou et al. 2008; Li et al. 2007; Fragouli et al. 2006].

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*, in which 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 a node will be corrupted. This will further affect other nodes and result in an epidemic propagation of corrupted packets 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 multihop 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. As a result, many such systems have been developed [Chachulski et al. 2007; Radunovic et al. 2007; Katti et al. 2005]. However, as recently shown in Dong et al. [2008], 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 when authentication mechanisms are used, such networks are vulnerable to insider attacks because 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, *intraflow* network coding and *interflow* 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: intraflow network coding systems mix packets within a single flow, while interflow network coding systems mix packets across multiple flows.

In this article, we focus on defense mechanisms against pollution attacks in intraflow network coding systems for wireless mesh networks. In existing intraflow coding systems [Chachulski et al. 2007; Park et al. 2006; Fragouli et al. 2006], intermediate nodes do not decode received packets, but use them to generate new coded packets. To prevent pollution attacks, 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 sourcegenerates 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 pollution attacks in intraflow coding systems use special-crafted digital signatures [Charles et al. 2006; Yu et al. 2008; Zhao et al. 2007; Li et al. 2006] or hash functions [Krohn et al. 2004; Gkantsidis and Rodriguez 2006], which allow intermediate nodes to verify the integrity of combined packets using their homomorphic properties. 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. Noncryptographic solutions have also been proposed [Ho et al. 2004; Jaggi et al. 2007; Wang et al. 2007]. These solutions either provide only a partial solution by detecting the attacks without any response mechanism [Ho et al. 2004], or add data redundancy at the source, resulting in throughput degradation proportional to the bandwidth available to the attacker [Jaggi et al. 2007; Wang et al. 2007].

We propose two practical schemes to address pollution attacks against intraflow 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 article to propose practical defenses against pollution attacks in wireless networks and to demonstrate their effectiveness in a practical system. Our main contributions are as follows.

—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 [Charles et al. 2006; Yu et al. 2008; Zhao et al. 2007; Li et al. 2006; Krohn et al. 2004] are able to achieve only less than 10% of the throughput that is typically available.

—We design DART (Delayed Authentication with Random Transformations), 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 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 one out of 65536 polluted packets to pass a first hop neighbor of the attacker, and one out of over 4 billion polluted packets to pass 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 further enhance both DART and EDART with an efficient attacker identification scheme, which isolates the attacker nodes quickly. The attacker identification scheme for EDART leverages the independence of symbols in the coding process and a novel *cross-examination* technique to efficiently identify attackers.

- We validate the performance and the overhead of our schemes with extensive simulations using a well-known network coding system for wireless networks (MORE [Chachulski et al. 2007]) and realistic link quality measurements from the Roofnet[1] experimental testbed, developed at MIT. The 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. Our schemes incur a bandwidth overhead of less than 2% of the system throughput and require approximately five digital signatures per second at the source. The attacker identification schemes can identify attackers within around one second of the attack, and incur only a small bandwidth overhead on the network.

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, DART and EDART. Section 7 proposes the attacker identification schemes for both DART and EDART. Section 8 demonstrates the impact of the attacks and the effectiveness of our defense mechanisms through simulations. Finally, Section 9 concludes the article.

2. RELATED WORK

2.1. 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. An exception to the paradigm of homomorphic hash or signatures is a recently proposed scheme [Agrawal and Boneh 2009] based on efficient homomorphic MACs. However, the scheme requires a key predistribution from a centralized trusted entity and its security decreases as the number of compromised nodes increases. In the following, we only consider homomorphic hash- or signature-based schemes which ensure security irrespective of the number of compromised nodes.

In hash-based schemes [Krohn et al. 2004; Gkantsidis and Rodriguez 2006], 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 Krohn et al. [2004] also has a high computational overhead. To overcome this limitation, Gkantsidis and Rodriguez [2006] proposed probabilistic batch verification in conjunction with a cooperative detection mechanism. This scheme was proposed for and works reasonably well in P2P 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. The "null keys" work

[1] http://pdos.csail.mit.edu/roofnet/doku.php.

[Kehdi and Li 2009] proposes to thwart pollution attacks by checking if coded packets belong to the subspace spanned by the native packets. This work targets network coding in P2P networks and assumes noncolluding attackers and topological properties such as path diversity. Both assumptions are unrealistic in wireless mesh networks.

Schemes based on digital signatures [Charles et al. 2006; Yu et al. 2008; Zhao et al. 2007; Li et al. 2006] 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 [Boneh et al. 2009] which achieves a 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, the schemes in Charles et al. [2006] and Boneh et al. [2009] rely on expensive bilinear maps, while those in Yu et al. [2008], Zhao et al. [2007] and Li et al. [2006] require a large number of modular exponentiations. Although the schemes proposed in Yu et al. [2008], Zhao et al. [2007], and Li et al. [2006] 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 Section 4, we argue in detail that cryptographic approaches have high overhead even under benign conditions, making them impractical for use in a wireless network.

2.2. Information Theoretic Approaches

One information-theoretic approach [Ho et al. 2004] 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 [Jaggi et al. 2007] 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 zero. 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. Wang et al. [2007] proposed 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.

3. SYSTEM AND ADVERSARIAL MODEL

3.1. System Model

We consider a general intraflow 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 subsequences called *generations*. Each generation consists of n packets and is disseminated independently to the receivers using network coding. We consider a setting in which network coding is performed at the routing layer (or higher).

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 $\mathbb{F}_q$, that is, 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 Chachulski et al. [2007], 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 article are column vectors.

This model is the general network coding framework proposed in Chou and Wu [2007] and fits all existing intraflow network coding systems known to us, including those in Chachulski et al. [2007], Park et al. [2006], Lin et al. [2008], Radunovic et al. [2007], and Cui et al. [2008]. 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, that is, *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 [Chachulski et al. 2007; Radunovic et al. 2007; Park et al. 2006] 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 [Perrig et al. 2002a], we assume that clocks in the network are loosely synchronized and that each node knows the upper bound on the clock skew between itself and the source, denoted as Δ. Several mechanisms to securely achieve such loose clock synchronization have beed provided in Perrig et al. [2002a, 2002b]. Other techniques [Sun et al. 2006a] reduce the synchronization error to the order of microseconds.

We also assume the existence of an end-to-end message authentication scheme, such as a digital signature or a 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 layers. 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 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 [Charles et al. 2006; Li et al. 2006; Zhao et al. 2007; Yu et al. 2008] and homomorphic hashes [Krohn et al. 2004; Gkantsidis and Rodriguez 2006]. Below we argue that the high computational cost of these schemes makes them impractical for wireless systems.

In the existing cryptographic-based schemes [Charles et al. 2006; Li et al. 2006; Zhao et al. 2007; Yu et al. 2008; Krohn et al. 2004; Gkantsidis and Rodriguez 2006], 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).

We argue that, 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 that the packet size is 1500 B and that each node in the system is equipped with a 3.4-Ghz Pentium IV processor, which performs around 250 modular exponentiations per second (according to OpenSSL 0.9.8e). Therefore, the destination can verify $\frac{250}{\gamma}$ packets per second and the throughput is $\tau = (1-\rho)\frac{250}{\gamma} \times 1500$ bytes/s, or equivalently,

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

which achieves the maximum value of 502 kb/s 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 502 kb/s, regardless of the actual link bandwidth, which can be much larger, for example, 11 Mb/s. 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 Section 8.3) show that, under typical network settings, the achievable throughput is around 50 kb/s, 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, for example, 20 or 32 bytes [Charles et al. 2006; Li et al. 2006; Zhao et al. 2007; Yu et al. 2008]. However, in all the practical network coding systems in wireless networks [Chachulski et al. 2007; Park et al. 2006; Radunovic et al. 2007], the symbol size used is much smaller, usually 1 byte. This is because arithmetic operations over a field are used extensively 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 [Perrig et al. 2002a], 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 G, 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.

Each forwarder node maintains two packet buffers, verified_set and unverified_set, that buffer the verified and unverified packets, respectively. Each forwarder combines only packets in the verified_set to code 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 authentic, the node rebroadcasts it to its neighbors. It then uses the checksum to verify the 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 linearly 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 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 packets pass the 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 can 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 attacker-injected 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 Section 5.3.

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 (i.e., batch verification).

As mentioned in the system model (Section 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 pseudorandom 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 pseudorandom 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:

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

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

If Equation (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 are identified efficiently using a binary search-like technique.

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 experimentally evaluated in Section 8.4.

5.3. Pipelining Across Generations

As discussed in Section 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 [Park et al. 2006; Radunovic et al. 2007] already incorporate pipelining for performance purposes 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, for example, MORE [Chachulski et al. 2007].

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, the maximum time skew in the network. 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 have shown 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 described in Section 8 have shown that, due to the small size of checksums, the overall bandwidth overhead remains small.

5.4. Security Analysis

We discuss below the security properties of DART by focusing on one generation. Pipelining across several active generations has no implication on this security analysis 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 that observes a checksum is to generate corrupted packets that will match the verification against that checksum at honest nodes. The key point of our scheme that prevents this attack 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. The following theorems quantify the security level of the checksum verification scheme with respect to two system parameters, q and b, where q is the field size used by network coding and b is the security parameter for the checksum generation as described in Section 5.2.

LEMMA 5.1. *In DART, all packets that are verified against a checksum packet are safe with respect to that checksum packet.*

PROOF. Since packet safety is based on relative timing, without loss of generality, we use the time at the source as the reference time. Denote by t_c the creation time of the checksum packet. For any coded packet that is verified against this checksum packet at any node, denote its creation time by t_g and its receiving time by t_r. Hence, $t_r > t_g$. We need to prove $t_g < t_c$.

When the packet is received by the node, let the local time at the node be t'_r and the time at the source be t_r. Since the maximum clock difference between the node and the source is Δ, we have $|t_r - t'_r| < \Delta$; hence, $t_r < t'_r + \Delta$. In the DART scheme, the timestamp in the checksum packet received by the node is t_c and a node only verifies a packet against this checksum packet if $t'_r < t_c - \Delta$. Thus, we have $t_g < t_r < t'_r + \Delta < t_c$. Hence, the packet is safe with respect to the checksum packet. □

THEOREM 5.2. *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* $\frac{1}{q^b}$.

PROOF. Let $\mathsf{CHK} = (\mathsf{CHK}_s(G), s, t)$ be a checksum for a generation G and $(\vec{c}, \vec{e})$ be a polluted packet, i.e., $\vec{e} \neq G\vec{c}$. We calculate the probability of the event that $(\vec{c}, \vec{e})$ passes the checksum verification, *i.e.*, the equation $\mathsf{CHK}_s(G)\vec{c} = H_s\vec{e}$ holds, as follows.

Let $\vec{e'}$ be the correct encoding for the code vector $\vec{c}$, that is, $\vec{e'} = G\vec{c}$. Therefore, $\vec{e'} \neq \vec{e}$. Let $H_s = [\vec{h_1}, \vec{h_2}, \ldots, \vec{h_m}]$ be the $b \times m$ random matrix generated from the seed s, where each $\vec{h_i}$ is a vector of b elements in $\mathbb{F}_q$.

Since both $(\vec{c}, \vec{e})$ and $(\vec{c}, \vec{e'})$ pass the checksum verification, we have $\mathsf{CHK}_s(G)\vec{c} = H_s\vec{e}$ and $\mathsf{CHK}_s(G)\vec{c} = H_s\vec{e'}$. Thus we have $H_s\vec{e} = H_s\vec{e'}$. Hence $H_s(\vec{e} - \vec{e'}) = 0$. Let $\vec{u} = (u_1, u_2, \ldots, u_m) = \vec{e} - \vec{e'}$, Since $\vec{e'} \neq \vec{e}$, we have at least some i, $1 \leq i \leq m$, such that $u_i \neq 0$. Without loss of generality, we assume $u_1 \neq 0$. Rewrite $H_s\vec{u} = 0$ as $\vec{h_1}u_1 + \vec{h_2}u_2 + \cdots + \vec{h_m}u_m = 0$. Since $u_1 \neq 0$, u_1 has a unique multiplicative inverse v_1 in $\mathbb{F}_q$ such that $u_1v_1 = 1$. We have

$$\vec{h_1} = -v_1(\vec{h_2}u_2 + \vec{h_3}u_3 + \cdots + \vec{h_m}u_m). \tag{2}$$

Therefore, given a fixed $\vec{h_2}, \vec{h_3}, \ldots, \vec{h_m}$ and $u_1, u_2, \ldots, u_m$, there exists a unique $\vec{h_1}$ that satisfies the above equation. Since the packet $(\vec{c}, \vec{e})$ is safe with respect to the checksum, the $\vec{h_i}$ vectors are generated at random after the u_i values are determined. Thus, although the attacker can control the values for u_i, the probability that Equation (2) holds is still equal to probability that the randomly selected $\vec{h_1}$ is the unique vector required. Since $\vec{h_1}$ consists of b symbols in $\mathbb{F}_q$ and each symbol was obtained using a pseudorandom function with output in $\mathbb{F}_q$, the probability of that to occur is $1/q^b$. □

THEOREM 5.3. *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* $\frac{1}{q^w} + \frac{1}{q^b}$.

Proof. We evaluate the probability that a set of packets containing polluted packets $E = \{(\vec{c_1}, \vec{e_1}), (\vec{c_2}, \vec{e_2}), \ldots, (\vec{c_l}, \vec{e_l})\}$ passes w independent batch checksum verifications as follows. Let $(\vec{c}, \vec{e})$ be a random linear combination of the packets in E with coefficients $u_1, \ldots, u_l$ selected uniformly at random from $\mathbb{F}_q$, that is, $(\vec{c}, \vec{e}) = (\sum_{i=1}^{l} u_i \vec{c_i}, \sum_{i=1}^{l} u_i \vec{e_i})$.

Let $E' = \{(\vec{c_1}, \vec{e'_1}), \ldots, (\vec{c_l}, \vec{e'_l})\}$ be the set of correctly coded packets with the same coefficients as in E, and $(\vec{c}, \vec{e'})$ be the linear combination of packets in E' obtained with the same coefficients $u_1, \ldots, u_l$, that is, $(\vec{c}, \vec{e'}) = (\sum_{i=1}^{l} u_i \vec{c_i}, \sum_{i=1}^{l} u_i \vec{e'_i})$. Since packets in E' are correctly coded packets, $(\vec{c}, \vec{e'})$ is also a correctly coded packet.

Let U_i be the event that $\vec{e} = \vec{e'}$ holds for the ith batch verification and V_i be the event that the packet $(\vec{c}, \vec{e})$ successfully passes the verification for the ith batch verification.

Let F_i be the event that E passes the ith batch verifications. Then, clearly, $U_i \subseteq F_i$ and $F_i = U_i \cup (V_i \cap \bar{U_i})$. Let $H = \bigcap_{1 \le i \le w} F_i$, that is, H is the event that E passes all w batch verifications. We evaluate the probability $P(H)$ that H occurs as follows.

$$\begin{aligned} P(H) &= P\left(\bigcap_{1\le i\le w} F_i\right) = P\left(\bigcap_{1\le i\le w} (U_i \cup (V_i \cap \bar{U_i}))\right) = P\left(\bigcap_{1\le i\le w} U_i \cup \bigcap_{1\le i\le w} (V_i \cap \bar{U_i})\right) \\ &\le P\left(\bigcap_{1\le i\le w} U_i\right) + P\left(\bigcap_{1\le i\le w} (V_i \cap \bar{U_i})\right) \le P\left(\bigcap_{1\le i\le w} U_i\right) + P(V_1 \cap \bar{U_1}) \\ &= \prod_{1\le i\le w} P(U_i) + P(V_1 | \bar{U_1}) P(\bar{U_1}) \le \prod_{1\le i\le w} P(U_i) + P(V_1 | \bar{U_1}), \end{aligned} \tag{3}$$

where the second to last part of the equation is because U_i's are independent events, since at each batch verification, the coefficients $u_1, \ldots, u_l$ are selected independently at random.

By Theorem 5.2, we have $P(V_1 | \bar{U_1}) = 1/q^b$. We now evaluate $P(U_i)$, which is the probability that the equation $\sum_{i=1}^{l} u_i (\vec{e_i} - \vec{e'_i}) = 0$ holds. Since at least one packet in E is polluted, we have that $\vec{e_i} \neq \vec{e'_i}$ for at least some i, $1 \le i \le l$. Without loss of generality, we assume $\vec{e_1} \neq \vec{e'_1}$. Therefore, we have that $e_{1j} \neq e'_{1j}$ for at least some j, $1 \le j \le m$. Again, without loss of generality, we assume $e_{11} \neq e'_{11}$. Let W be the event that $\sum_{i=1}^{l} u_i (e_{i1} - e'_{i1}) = 0$. Let $\vec{v_i} = \vec{e_{i1}} - \vec{e'_{i1}}$. We have $\sum_{i=1}^{l} u_i v_i = 0$. Since $v_1 \neq 0$, v_1 has a unique multiplicative inverse β_1 in $\mathbb{F}_q$; thus, we have $u_1 = -\beta_1 (\sum_{i=2}^{l} u_i v_i)$. Since u_1 is randomly selected from $\mathbb{F}_q$, given fixed $u_2, \ldots, u_l$ and $v_1, \ldots, v_l$, the probability that equation $u_1 = -\beta_1 (\sum_{i=2}^{l} u_i v_i)$ holds is $1/q$, that is, $P(W) = 1/q$. Since $U_i \subseteq W$, we have $P(U_i) \le P(W) = 1/q$. Therefore, by Equation (3) we have $P(H) \le \frac{1}{q^w} + \frac{1}{q^b}$. □

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 (Section 8) 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.

Checksum dropping attack. 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 one or two hops).

Size of the security parameters. As shown in Theorems 5.2 and 5.3, 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.

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 and switch 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

ALGORITHM 1: EDART Scheme Executed by Each Forwarder Node

```
Executed at system initialization
  $C_v = 0$; forward_set = $\emptyset$; delay_set = $\emptyset$
Executed on receiving packet $p$
  if ($C_v > 0$ or $h_v \geq \delta$) then add $p$ to delay_set
  else add $p$ to forward_set
Executed to output a packet
  Select a subset of packets from forward_set to form a coded packet as required by the
    particular network coding system.
  Set $h_{\max}$ to be the maximum $h_v$ in the selected packets
  For the coded packet, set $h_v = h_{\max} + 1$
Executed on receiving checksum $(\mathsf{CHK}_s(G), s, t)$
  Verify all unverified safe packets in both forward_set and delay_set against $\mathsf{CHK}_s(G)$
  Set $h_v = 0$ for all verified packets
  if there exist invalid packets that failed verification then
      Set $h_{\min}$ to be the minimum $h_v$ in all packets that failed verification
      $C_v = C_v + \alpha(1 - \frac{h_{\min}}{\delta})$
  else if $C_v > 0$ then
      $C_v = C_v - 1$
```

near an attacker tend to verify packets, while nodes farther away tend to forward packets without delay.

The detailed pseudocode 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 predetermined 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 zero). 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 predetermined system parameter. For all packets that pass the verification, their h_v field is reset to zero.

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 attack impact averaged over time.

—*Pollution success frequency*. The frequency of pollution attacks with scope greater than 1. 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 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, the attacker sets $h = \delta - 1$, resulting in the maximum success frequency of $1/(\alpha(1 - \frac{\delta - 1}{\delta}) + 1) < \delta/\alpha$. □

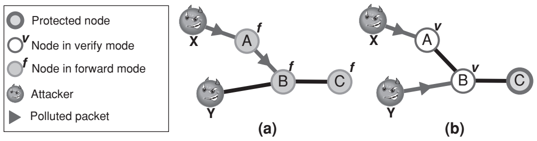


Fig. 1. (a) Attacker X attacks first and nodes A and B receive polluted packets; (b) in the following time interval, A and B switch to the verify mode, and attacker Y starts attacking. Y's polluted packets are immediately dropped by node B and further nodes (such as node C) are protected. Thus, the strength 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 $2 \le i \le \delta - h - 1$) have unnecessary delay of $\alpha(1 - \frac{h+i}{\delta})$ time intervals. Nodes further than $\delta - h - 1$ hops have no unnecessary delay.*

PROOF. Since the h_v value is incremented at each hop, nodes that are i hops away from the attacker (with $2 \le i \le \delta - h - 1$) 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. □

Multiattacker 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 an honest node is in the pollution scope of only one attacker. In this case, we can view the network subdivided into smaller areas, each of which contain 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 Figure 1, 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 Section 8 confirm that EDART remains effective against pollution attacks in the presence of multiple attackers.

6.3. Selection of δ and α

Parameter δ is defined as the number of hops after which a coded packet is always verified. Parameter α is defined as 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 resiliency of the system. 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.

7. ATTACKER IDENTIFICATION

DART and EDART focus on detecting and dropping polluted packets. Although this effectively limits the impact of pollution attacks, for a practical solution it is also important to be able to identify the attacker nodes. First, the ability to identify attacker nodes allows the source node to exclude attackers on the data delivery path and make optimal selection of forwarder nodes among only honest nodes. Otherwise, the attacker nodes may be selected on the critical path, rendering the flow under their complete control. Second, excluding attacker nodes from the data delivery path eliminates further packet pollution from the identified attacker nodes. This is particularly useful for EDART because, in the absence of packet pollution, nodes will operate in forward-only mode, which improves performance. Finally, the identification of attacker nodes also allows for physical intervention to recover compromised nodes, for example, by reinstalling software on them. In this section, we enhance both the DART and EDART schemes to efficiently identify pollution attacker nodes.

7.1. Assumptions

In order to be able to attribute the packet pollution to a certain node, it is crucial to have the nonrepudiation property on packets. Thus we assume that each forwarder node signs every coded packet it generates. We further assume the existence of a reliable end-to-end communication path between every pair of nodes. Reliable end-to-end communication in wireless networks has been a subject of extensive study with numerous proposals [Hu et al. 2002; Awerbuch et al. 2008; Guerrero et al. 2002], any of which may be used with our protocol.[2] Last, we assume that there is no Sybil attack, in which a single attacker node owns multiple (bogus) identities and their associated credential information.

7.2. DART-AI: DART with Attacker Identification

Since in DART each node verifies packets before using them for coding, honest nodes will only forward valid packets except for a small false-negative probability in the checksum verification scheme. Thus we propose the attacker identification scheme for DART (DART-AI) as follows: when a node receives a corrupted packet that does not pass the checksum verification, it reports the sender of the packet as a pollution attacker node to the source node along with the corrupted packet itself as a proof. The source node, on receiving such a report, checks that the packet reported is indeed corrupted and is signed by the reported attacker node. If so, the source regards the reported node as an attacker node; otherwise, the reporting node is regarded as an attacker node.

7.2.1. Security Analysis. Since there is a false-negative probability in the checksum verification, an honest node can also forward corrupted packets and, hence, be mistakenly identified as an attacker node. Conversely, an attacker node whose corrupted packets happen to pass the checksum verification at honest nodes can escape from being identified. Below we analyze such false positive and false negative probability of the attacker identification scheme for DART.

False positive probability. Let E_{fp} be the event that an honest node is mistakenly identified as an attacker node and E_1 the event that the node accepts a corrupted packet as a valid one. Since E_{fp} occurs if the node accepts a corrupted packet as a valid one and the packet produced by the node is correctly identified by other nodes as corrupted, we have

$$Pr(E_{fp}) \leq Pr(E_1).$$

[2] Since reliable communication is required only after an attack is positively identified, even a straightforward implementation using flooding can be used.

Let k be the number of corrupted packets received by the node; then the probability that any of these packets are accepted as valid by the node is

$$Pr(E_1) = 1 - (1 - 1/q^b)^k.$$

Thus we have $Pr(E_{fp}) \leq 1 - (1 - 1/q^b)^k$. Assuming a typical setting of $q = 256$ and $b = 2$, the false positive probability is less than 0.002 even when k is as large as 100.

False negative probability. Let E_{fn} be the event that a pollution attacker is not identified after injecting a *single* polluted packet. Thus $Pr(E_{fn})$ is the lower-bound on the probability that a pollution attacker goes undetected. Let c be the number of checksums that the injected corrupted packet is verified against at an honest node.[3] The attacker is not identified only if its injected packet passes all c different checksum verifications. Thus

$$Pr(E_{fn}) = (1/q^b)^c.$$

In the worst case, when the injected corrupted packet is verified against only one checksum,[4] we have $Pr(E_{fn}) \leq 1/q^b$. Again, with a typical setting of $q = 256$ and $b = 2$, we have $Pr(E_{fn}) \leq 1/2^{16}$.

7.3. EDART-AI: EDART with Attacker Identification

In EDART, an honest node may forward many corrupted packets because in the forward mode it uses the received packets for coding without verifying them. Thus, unlike in DART, we cannot simply accuse any node that forwards corrupted packets as an attacker. Instead, we adopt a traceback strategy that traces backward through the causal relationship of the packet pollution until an attacker is identified.

The strategy relies on the observation that a corrupted packet produced by an honest node with an unverified hop count field h_{v1} is always caused by corrupted packets received by the node with an unverified hop count field $h_{v2} < h_{v1}$. Thus, to identify the attacker node, the source iteratively queries each node that produces corrupted packets for its input packets that have a smaller h_v field than the corrupted packet produced by the node. In this process, an honest node is always able to provide one such packet as a proof of its coding correctness (thus being declared innocent), while the attacker node who injects corrupted packets is not able to do so. Thus, by tracing through the packet causal relationship in the reverse direction of the h_v field, an attacker can eventually be identified.

A naive implementation of the above strategy would require each queried node to return to the source an entire corrupted input packet as a proof of its innocence. Instead, we propose an efficient symbol-level traceback technique that requires the queried nodes to only return a single symbol in the packet.[5] The symbol-level traceback strategy relies on the observation that each symbol in a packet is coded independently of each other. A corrupted symbol at some position of a coded packet is always caused by corrupted symbols at the same position of some input packets. Thus the source only needs to trace the causal relationship in the generation of a corrupted symbol in a packet in order to identify the attacker node. One challenge of symbol-level traceback is that an attacker node may claim an arbitrary input symbol being from any node in the network, as symbols are not signed individually by forwarder nodes. To address

[3] As noted in Section 5.4, the checksum verification has no false positives; thus, an honest node can use newly received checksums to verify packets in both unverified_set and verified_set to further reduce the false negative probability of checksum verificatoin. The use of batch verification and low overhead of checksum computation ensure that the overall overhead remains low.

[4] If it is not verified at all, the packet will not be used for coding, and thus will not cause any pollution.

[5] This is a symbol used in network coding as defined in Section 3.1, which usually has a length of 1 byte.

this, we incorporate a *cross-examination* technique which checks the consistency of the claimed value for a symbol with the claimed sender of the packet. If the claimed packet sender disagrees on the symbol value, then either the claiming node or claimed packet sender is lying. The source node can can resolve this dispute by querying the claiming node for the complete packet containing the symbol as signed by the claimed sender node.

In the following, we first introduce some notations and formally define a corrupted symbol, and then describe the details of the symbol-level traceback procedure.

7.3.1. Notations. As defined in Section 3.1, each packet $\vec{p_i}$ in a generation is viewed as a column vector of m elements over $\mathbb{F}_q$, that is, $\vec{p_i} = (p_{i1}, p_{i2}, \ldots, p_{im})^T$ with $p_{ij} \in \mathbb{F}_q$. A generation G of n packets is viewed as a matrix $G = [\vec{p}_1, \ldots, \vec{p}_n]$ with each packet in the generation as a column in the matrix. We further define $\vec{r_j}$ to be the jth row of the matrix G, that is, $\vec{r_j} = (p_{1j}, p_{2j}, \ldots, p_{nj})$. Given a corrupted coded packet $p = (\vec{c}, \vec{e})$ with $\vec{c} = (c_1, c_2, \ldots, c_n)$ and $\vec{e} = (e_1, e_2, \ldots, e_m)$, we say a symbol e_j is a corrupted symbol if $e_j \neq \sum_{i=1}^{n} c_i p_{ij}$, or equivalently, $e_j \neq \vec{c} \cdot \vec{r_j}$, where "$\cdot$" represents the vector dot product. Thus, given $\vec{r_j}$ and a coded packet $(\vec{c}, \vec{e})$, one can verify whether a symbol e_j in $\vec{e}$ is corrupted. Clearly, a corrupted packet always contains at least one corrupted symbol.

7.3.2. Scheme Details. The symbol-level traceback starts when the source receives a pollution report with a corrupted coded packet $(\vec{c}, \vec{e})$. To start the traceback, the source first locates one corrupted symbol in the corrupted coded packet.[6] Let j be the position of the corrupted symbol in $\vec{e}$. The source maintains the following state: the current prover node P, the previous prover node P', the current value for the h_v field, denote as h, and the value e_j of the corrupted symbol at the jth position of a packet produced by the prover node P as claimed by the previous prover node P'.

At first, the source node initializes the current prover node P to be the node that reports the pollution and h to be the h_v field of the reported polluted packet. Both P' and e_j are initialized to be a special *null* value indicating that it is not used. In each round, the source sends to the current prover node a query that contains a tuple $T_q = (\vec{r_j}, j, h, e_j)$, where $\vec{r_j}$ is the jth row of the matrix G. The value e_j in the query tuple T_q is for cross-examination of the symbol between the previous prover node and the current prover node.

On receiving a query tuple $T_q = (\vec{r_j}, j, h, e_j)$, if e_j is not null, the node first validates the claim made by the previous prover node on e_j by checking if it has generated a corrupted packet p whose h_v field is h and whose jth symbol is corrupted and has value e_j. If any of these checks fail, the node returns a response indicating the cross-examination failure. Otherwise, the node checks all the coded packets in its buffer with the h_v field smaller than h for the correctness of their jth symbol. To do so, for a coded packet $(\vec{c}, \vec{e})$, it checks if the equation $e_j = \vec{c} \cdot \vec{r_j}$ holds. An honest node will always be able to find a corrupted packet whose jth symbol is corrupted and with $h_v < h$. In response to the query, the node returns a response tuple $T_r = (e_j, h_v, n)$, where e_j and h_v are the jth symbol and the h_v field of the corrupted packet, respectively, and n is the identifier of the node that sent the packet.

If the source fails to receive a response from the current prover node P after a time out, it identifies P as an attacker and terminates the traceback procedure. If the received response indicates a cross-examination failure, then either the current prover node P or the previous prover node P' is lying and can be identified as an attacker. To find out which one, the source queries the previous prover node P' for the packet signed by P and whose jth symbol is corrupted and has value e_j. If node P', which

[6] The source is always able to do so, as it has all the plain packets for the respective generation.

should have stored the packet in its buffer, can return the queried packet, then node P is identified as the attacker. Otherwise, node P' is identified as the attacker.

If the source successfully receives a response tuple $T_r = (e_j, h_v, n)$ with e_j being corrupted and $h_v < h$, it repeats the traceback on node n by setting P' to be P, P to be n, h to be h_v, and e_j to be the e_j in T_r. This process repeats until an attacker is identified.

7.3.3. Security Analysis. Since the correctness of plain packets is ensured based on end-to-end authentication, packet corruption is always detected at some honest node either due to the failure of checksum verification or when the packet is decoded at the receiver node. Therefore, for every packet pollution attack, a traceback procedure is always invoked. We show next that every invocation of the traceback procedure will result in an attacker node being identified with high probability. Therefore, we can conclude that for every occurrence of a pollution attack, an attacker is identified with high probability.

LEMMA 7.1. *On each traceback procedure, an attacker is identified correctly with the probability of* $1 - 1/q^b$*, and the probability that an honest node is mistakenly identified as an attacker node (i.e., the false positive probability) is* $1/q^b$.

PROOF SKETCH. An honest node will only be identified as an attacker when it forwards a corrupted packet with its h_v field reset to zero. This only occurs when the node mistakenly classifies a corrupted packet as a correct packet, because an honest node only resets the h_v field after performing a checksum verification. By Theorem 5.2, the false negative probability of the checksum verification is $1/q^b$; thus we have the honest node being mistakenly identified as an attacker node is also $1/q^b$.

Since the traceback procedure always identifies some node as an attacker node, we have that the probability that an attacker node is correctly identified is $1 - 1/q^b$. □

7.3.4. Overhead Analysis. We analyze the computation, bandwidth, and storage overhead of the traceback procedure in EDART-AI. The computation overhead involves only signatures operations (i.e., creation and verification) for the query and response messages. Since the number of traceback rounds is upper-bounded by δ (the maximum value for h_v), the maximum number of signature operations to identify an attacker is 4δ.

For the bandwidth overhead, each query packet $(\vec{r_j}, j, h, e_j)$ is of size $|\vec{r_j}| + |j| + |h| + |e_j| = (n+1)q + 2$ bytes, where n is the number of packets in a generation, q is the packet size, and $|x|$ denotes the size of x. A response packet (e_j, h_v, n) is of size $q + 3$ bytes, assuming a node identifier needs 2 bytes. Since the maximum number of traceback round is δ, the maximum bandwidth overhead incurred is $((n+2)q + 5)\delta$. With a realistic setting of $n = 32, q = 1, \delta = 8$, the maximum bandwidth overhead of a traceback procedure is less than as 320 bytes.

In order to respond to traceback queries and to perform cross-examination, honest nodes need to store both the received input coded packets and output coded packets until a generation is successfully delivered to the receiver. A node is already required to store the input packets for the purpose of network coding. Thus the storage overhead induced by the traceback procedure is only the storage of the output packets. As validated by our experiments, for each generation a node typically only sends tens of packets. Thus the storage overhead of the traceback procedure is small.

8. EXPERIMENTAL EVALUATION

We first 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 [Chachulski et al. 2007], 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 its source code is publicly available. We use the real-world link quality measurements from Roofnet,[7] an experimental 802.11 b/g mesh network in development at MIT. Roofnet has also been widely used in other research papers [Radunovic et al. 2007; Couto et al. 2003; Aguayo et al. 2004; Bicket et al. 2005; Biswas and Morris 2004] as a representative wireless mesh network testbed.

8.1. Experimental Methodology

Simulation setup. Our experiments were performed with the Glomosim simulator[8] configured with 802.11 as the MAC layer protocol. We used realistic link quality measurements for the physical layer of Glomosim to determine the link loss rate. Specifically, we used the real-world link quality trace data from the Roofnet testbed. The network consists of 38 nodes, and the topology and link qualities are shown in Dong et al. [2009], Figure 3. The raw bandwidth is 5.5 Mb/s.

We assumed the clocks of nodes in the network were loosely synchronized, with the maximum clock drift between any node and the source being $\Delta = 100$ ms.[9] We used Elliptic Curve DSA signature (ECDSA) with 160-bit key size[10] and simulates delays to approximate the cryptographic performance for operations on a 3.4-GHz Intel Pentium 4 processor.

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

MORE setup. We used the default MORE setup [Chachulski et al. 2007], with a finite field for network coding of $\mathbb{F}_{2^8}$, generation size n of 32 packets, and packet size of 1500 B.

Attack scenario. We varied the number of attackers from one to 10 (out of a total of 38 nodes). We selected attackers only among forwarder nodes and the neighbors of forwarder nodes, because only these nodes can cause packet pollution. If the total number of such nodes was less than the specified number of attackers, we selected all of them as attackers.

The attackers injected polluted packets, but followed 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. PI captures the frequency of pollution attacks executed by an attacker. We varied PI to examine the impact of different levels of attack intensity.

DART and EDART setup. We set the checksum parameter $b = 2$, which resulted in a checksum size of 64 bytes. The source broadcast a checksum packet after broadcasting every 32 data packets.[11] We used the pipelining technique described in Section 5.3, with the pipeline size of 5. For EDART, we used $\delta = 8$ and $\alpha = 20$. These parameters

[7] See footnote 1 for web address where this is available.

[8] `http://pcl.cs.ucla.edu/projected/glomosim`.

[9] Current secure clock synchronization schemes [Sun et al. 2006a, 2006b] 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.

[10] Equivalent to the security level of RSA with 1024-bit key size.

[11] 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.

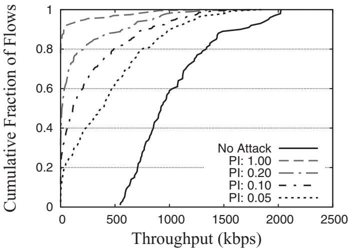


Fig. 2. The throughput CDF in the presence of a single attacker for various pollution intensities (PIs). Even when the attacker injects only one polluted packet every 20 received packets (PI = 0.05%), the impact of the attack is still very significant.

were experimentally selected to suit the small scale of the network and to balance between overhead, throughput, and latency.

As a baseline, we used a hypothetical ideal defense scheme referred to as *Ideal*, in which nodes detect polluted packets with zero cost and immediately drop them. Note that, under benign conditions, the *Ideal* scheme behaves the same as the original MORE protocol. We compared our schemes with *Ideal* using the same pipeline size to examine the latency caused by delayed packet verification.

DART-AI and EDART-AI setup. The experiments for DART-AI and EDART-AI were based on the settings for DART and EDART as described above. For the attacker identification, each packet was signed using the ECDSA signature scheme as described previously. We assumed the source would avoid the attacker nodes by reselecting the forwarding node set once the attacker nodes were identified. As a baseline to evaluate the performance of DART-AI and EDART-AI, we also defined a hypothetical ideal defense scheme with attacker identification referred to as *Ideal-AI* in which polluted packets are always filtered and the source node knows the attacker nodes without any overhead. Thus, in *Ideal-AI*, the packet forwarding process is the same as if the attacker nodes are taken out from the network.

Metrics. In our experiments, we measured the throughput, latency, and overhead (bandwidth and computation) of our defense schemes. *Throughput* was measured as the average receiving rate at which the destination received data packets (after decoding). *Latency* was measured as the delay in receiving the first packet (decoded) at the destination. For DART and EDART, the only bandwidth overhead incurred by our scheme was the dissemination of checksum packets. We measured 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 microbenchmark below), we measured the *computational overhead* as the number of digital signatures per second performed by the source for signing checksum packets. The overhead measurement did not include the overhead due to time synchronization. Similarly, for DART-AI and EDART-AI, we measured both the bandwidth and computation overhead due to packet signatures and the attacker identification process.

8.2. Impact of Pollution Attacks

To demonstrate the severity of pollution attacks on network coding, we evaluated the impact of the attack conducted by a *single attacker*. Figure 2 shows the throughput of MORE in a network with only one attacker with various pollution intensities. In the no attack case, we observe that all the flows have a throughput greater than 500 kb/s, with median at around 1000 kb/s. In the attack case with pollution intensity

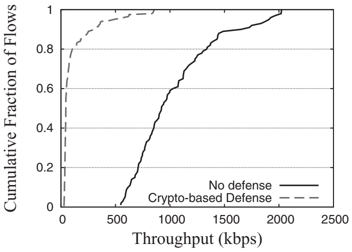


Fig. 3. Impracticality of previous work: throughput CDF of original MORE and of MORE with cryptographic-based defense in a benign network. Even when no attack takes place, previous defense schemes have a very high overhead that makes them impractical.

of 1, the throughput of around 80% of all flows goes to zero, and 97% of all flows have throughput less than 500 kb/s. 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 500 kb/s. Therefore, we conclude that pollution attacks are extremely detrimental to system performance, even when performed very infrequently and by only one attacker.

8.3. Limitations of Previous Solutions

We used 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 accounted for computational overhead at the intermediate nodes, and ignored 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 discounted such a large overhead of network coding for these schemes. Finally, we used 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 Section 5.3 to further boost the performance of such schemes.

Figure 3 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, for example, 80% of the flows have throughput 100 kb/s or less[12] and the median throughput degrades by 96%. Hence we conclude they are impractical for wireless mesh networks.

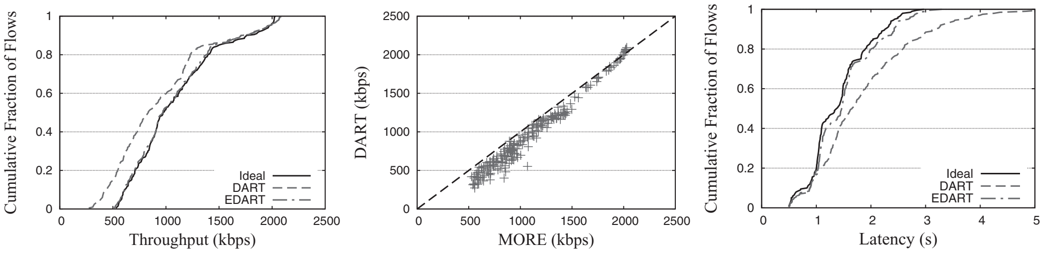
8.4. Evaluation of DART and EDART

We now evaluate the performance of our proposed defense schemes, DART and EDART. We first performed microbenchmarks to evaluated the computational cost of checksum generation and verification. We then evaluate the performance of our defense schemes

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

Table I. Computational Cost for Checksum Generation and Verification for Different Checksum Sizes

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 (for 32 packets) (ms)	0.492	1.319	2.458



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

Fig. 4. The throughput and latency of DART and EDART under benign case.

for benign networks and for networks under various pollution attack intensities. Finally, we examined their bandwidth and computational overhead.

Microbenchmarks. We evaluate the computational cost of checksum generation and verification on a computer with a 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 I summarizes the results.

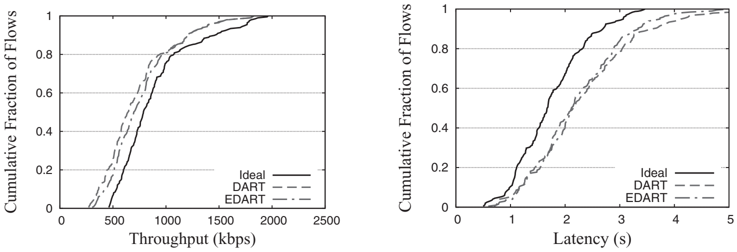
Benign networks. Figure 4 shows the throughput and latency of our schemes in a benign network with no attackers, as compared to the MORE protocol. In Figure 4(a), we see that DART incurs some throughput degradation (around 9% degradation when comparing median throughputs), whereas EDART incurs almost no degradation.

Figure 4(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 Figure 4(c), we observe a similar pattern for the latency of DART and EDART. DART incurs an additional 0.4 s 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.

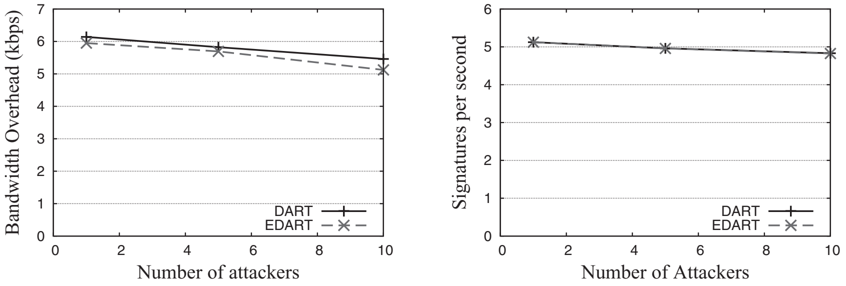
In summary, when no attacks take place, both of our schemes have throughput over 20 times higher than cryptographic-based schemes and cause minimal degradation on system performance. The performance of EDART is almost identical to the *Ideal* scheme.

Networks under attack. We examine the effectiveness of our defense against different number of pollution attackers. Figures 5(a) and 5(b) show the throughput and latency CDF of DART and EDART for the case of five attackers with pollution intensity of 0.2. From Figure 5(a), we see that both DART and EDART achieve a throughput close to the *Ideal* scheme. EDART achieves an even higher throughput than DART, especially



(a) Throughput CDF (b) Latency CDF

Fig. 5. The throughput and latency CDF of DART and EDART with five pollution attackers.



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

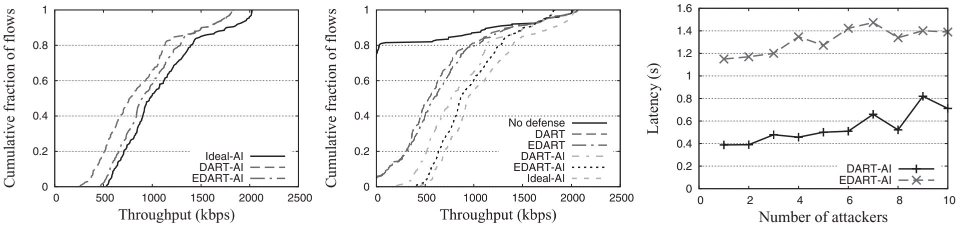
Fig. 6. Bandwidth and computational overhead of DART and EDART.

for low throughput flows, as these flows typically traverse many forwarder nodes. In Figure 5(b) we see that the median latency increase of DART and EDART over the *Ideal* scheme is around 0.5 s. 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 experimented with the cases of one and 10 attackers and other pollution intensities, all of which have results similar to the case of five pollution attackers 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 that of the *Ideal* scheme.

Overhead. Figures 6(a) and 6(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, 5.5 kb/s per forwarder node and five 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 5.5 kb/s is about 0.6% of the throughput achieved by the system on average.



(a) Throughput under benign network (b) Throughput under 5 random attackers (c) Attacker identification latency

Fig. 7. Throughput and attacker identification latency for the schemes with attacker identification.

It may also be counterintuitive that both the bandwidth and computational overhead decrease slightly when the number of attackers increases. The explanation is that the frequency with which 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.

8.5. Evaluation of DART-AI and EDART-AI

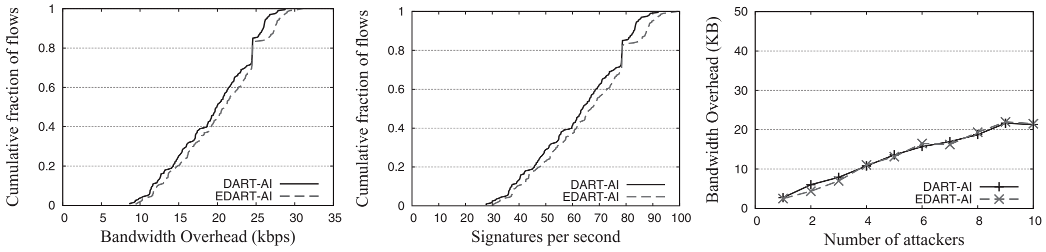
We evaluated the performance and overhead of DART-AI and EDART-AI. We first evaluated the throughput of the schemes under benign network environment. Then we examined the benefit of attacker identification when the network was under attack. Finally, we evaluated both the proactive and reactive overhead of attacker identification.

Benign networks. Figure 7(a) shows the throughput of the defense schemes with attacker identification in benign networks. The throughput of both DART-AI and EDART-AI is very close to that of the *Ideal-AI* defense scheme. Compared to the defense schemes without attacker identification (shown previously in Figure 4(a)), the degradation in throughput due to the additional overhead of attacker identification is very small.

Network under attack. Figure 7(b) shows the throughput of the defense schemes with and without attacker identification when the network contains five randomly placed attackers that perform both packet pollution and packet dropping attacks. In other words, the attackers inject polluted packets and do not forward any correct packets. We see that when the attackers refuse to forward any correct packets, DART and EDART suffer around 40% performance degradation in median. This is because although polluted packets are effectively filtered in DART and EDART, the presence of attackers in the forwarding node set causes a suboptimal packet forwarding compared to the case when forwarding nodes are selected only among honest nodes. By identifying and avoiding attacker nodes, both DART-AI and EDART-AI deliver a performance similar to that of the *Ideal-AI* defense scheme.

A key factor for the effectiveness of an attacker identification scheme is the latency of attacker identification, which measures the duration between the attack execution time and the attacker identification time. As illustrated in Figure 7(c), the latency of attacker identification for both DART-AI and EDART-AI remains at a stable level of around 0.5 s and 1.2 s, respectively. The main reason for the higher attacker identification latency in EDART-AI is the traceback process, which is not necessary in DART-AI.

Overhead. The overhead of DART-AI and EDART-AI consists of a proactive component and a reactive component. The proactive overhead involves the bandwidth and computation overhead associated with the packet signatures, while the reactive overhead is due to the overhead for identifying attackers. Figures 8(a) and 8(b) show the proactive bandwidth and computation overhead of the two attacker identification schemes. The proactive bandwidth overhead for both DART-AI and EDART-AI is



(a) Proactive bandwidth overhead (b) Proactive computation overhead (c) Reactive bandwidth overhead

Fig. 8. Computation and bandwidth overhead of the attacker identification schemes.

similar, ranging from 10 kb/s to 30 kb/s. The proactive computation overhead of the two schemes is also similar, ranging from 30 signatures per second to 90 signatures per second.

Figure 8(c) shows the reactive bandwidth overhead of DART-AI and EDART-AI, measured as the total number of bytes delivered for identifying all the attackers; in the network. The reactive bandwidth overhead increases linearly with the number of attackers; however, the total overhead incurred is only around 20 kB even for 10 attackers in the network. We also observe that EDART-AI incurs a similar level of bandwidth overhead as DART-AI, showing that the bandwidth overhead of the traceback procedure in EDART is small.

9. CONCLUSION

In this article, we present two new and practical defense schemes, DART and EDART, against pollution attacks in intraflow 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. We also propose enhancements on DART and EDART that allow efficient attacker identification and isolation. 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 the following.

—The effect of pollution attacks is devastating in mesh networks using intraflow 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.

—Both DART and EDART can effectively filter out polluted packets and restore the network performance to a level similar to a hypothetical *Ideal* defense scheme. The attacker identification schemes for both DART and EDART can identify attackers within around 1 s of the attack, thus allowing the selection of attacker-free paths and further improving the performance. All of our schemes incur a small overhead in terms of both bandwidth and computation.

REFERENCES

Agrawal, S. and Boneh, D. 2009. Homomorphic Macs: Mac-based integrity for network coding. In *Proceedings of the International Conference on Applied Cryptography and Network Security*.

Aguayo, D., Bicket, J., Biswas, S., Judd, G., and Morris, R. 2004. Link-level measurements from an 802.11b mesh network. *SIGCOMM Comp. Comm. Rev. 34,* 4, 121–132.

Ahlswede, R., Cai, N., Li, S.-Y., and Yeung, R. 2000. Network information flow. *IEEE Trans. Inform. Theor. 46,* 4, 1204–1216.

Awerbuch, B., Curtmola, R., Holmer, D., Nita-Rotaru, C., and Rubens, H. 2008. ODSBR: An on-demand secure Byzantine resilient routing protocol for wireless ad hoc networks. In *ACM Trans. Info. Syst. Sec. 10*, 4, Article 6.

Bicket, J., Aguayo, D., Biswas, S., and Morris, R. 2005. Architecture and evaluation of an unplanned 802.11b mesh network. In *Proceedings of the ACM International Conference on Mobile Computing Networking.*

Biswas, S. and Morris, R. 2004. Opportunistic routing in multi-hop wireless networks. *SIGCOMM Comp. Comm. Rev. 34,* 1, 69–74.

Boneh, D., Freeman, D., Katz, J., and Waters, B. 2009. Signing a linear subspace: Signature schemes for network coding. In *Proceedings of the International Conference on Public Key Cryptography.*

Chachulski, S., Jennings, M., Katti, S., and Katabi, D. 2007. Trading structure for randomness in wireless opportunistic routing. In *Proceedings of the ACM SIGCOMM Data Communications Festival.*

Charles, D., Jain, K., and Lauter, K. 2006. Signatures for network coding. In *Proceedings of the Annual Conference on Information Sciences and Systems.*

Chou, P. and Wu, Y. 2007. Network coding for the Internet and wireless networks. *IEEE Signal Process Mag. 24*, 77–85.

Chou, Y. W. P. A. and Kung, S.-Y. 2005. Minimum-energy multicast in mobile ad hoc networks using network coding. *IEEE Trans. Comm. 53*, 11, 1906–1918.

Couto, D. S. J. D., Aguayo, D., Bicket, J., and Morris, R. 2003. A high-throughput path metric for multi-hop wireless routing. In *Proceedings of the ACM Annual Conference on Mobile Computing and Networking.*

Cui, T., Chen, L., and Ho, T. 2008. Energy efficient opportunistic network coding for wireless networks. In *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies.*

Dana, A. F., Gowaikar, R., Palanki, R., Hassibi, B., and Effros, M. 2006. Capacity of wireless erasure networks. *IEEE Trans. Inform. Theor. 52*, 3, 789–804.

Deb, S. and Medard, M. 2006. Algebraic gossip: A network coding approach to optimal multiple rumor mongering. *IEEE Trans. Inform. Theor. 52*, 6, 2486–2507.

Dimakis, A. G., Godfrey, P. B., Wainwright, M. J., and Ramchandran, K. 2007. The benefits of network coding for peer-to-peer storage systems. In *Proceedings of the Workshop on Network Coding, Theory, and Applications*.

Dong, J., Curtmola, R., and Nita-Rotaru, C. 2009. Practical defenses against pollution attacks in intra-flow network coding for wireless mesh networks. In *Proceedings of the 2nd ACM Conference on Wireless Network Security*.

Dong, J., Curtmola, R., Sethi, R., and Nita-Rotaru, C. 2008. Toward secure network coding in wireless networks: Threats and challenges. In *Proceedings of the Fourth Workshop on Secure Network Protocols.*

Effros, M., Ho, T., and Kim, S. 2006. A tiling approach to network code design for wireless networks. In *Proceedings of the IEEE Information Theory Workshop*.

Fragouli, C. and Markopoulou, A. 2005. A network coding approach to overlay network monitoring. In *Proceedings of the Annual Allerton Conference on Communication Control and Computing.*

Fragouli, C. and Markopoulou, A. 2006. Network coding techniques for network monitoring: A brief introduction. In *Proceedings of the International Zurich Seminar on Communications.*

Fragouli, C., Widmer, J., and Le Boudec, J.-Y. 2006. A network coding approach to energy efficient broadcasting: From theory to practice. In *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies*.

Gkantsidis, C. and Rodriguez, P. 2005. Network coding for large scale content distribution. In *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies*.

Gkantsidis, C. and Rodriguez, P. 2006. Cooperative security for network coding file distribution. In *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies*.

Guerrero Zapata, M. and Asokan, N. 2002. Securing Ad hoc Routing Protocols. In *Proceedings of the ACM Workshop on Wireless Security* (*WiSe02*). 1–10.

Ho, T. 2006. On constructive network coding for multiple unicasts. In *Proceedings of the Annual Allrton Conference on Communication Control and Computing*.

Ho, T., Leong, B., Chang, Y.-H., Wen, Y., and Koetter, R. 2005. Network monitoring in multicast networks using network coding. In *Proceedings of the IEEE International Symposium on Information Theory*.

HO, T., LEONG, B., KOETTER, R., MEDARD, M., EFFROS, M., AND KARGER, D. 2004. Byzantine modification detection in multicast networks using randomized network coding. In *Proceedings of the IEEE International Symposium on Information Theory*.

HOU, I.-H., TSAI, Y.-E., ABDELZAHER, T., AND GUPTA, I. 2008. Adapcode: Adaptive network coding for code updates in wireless sensor networks. In *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies*.

HU, Y.-C., PERRIG, A., AND JOHNSON, D. B. 2002. Ariadne: A secure on-demand routing protocol for ad hoc networks. In *Proceedings of the ACM Annual International Conference on Mobile Computing Networking.*

JAGGI, S., LANGBERG, M., KATTI, S., HO, T., KATABI, D., AND MEDARD, M. 2007. Resilient network coding in the presence of byzantine adversaries. In *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies*.

JAIN, K. 2005. On the power (saving) of network coding. In *Proceedings of the Annual Allerton Conference on Communication Control and Computing.*

JIN, J., HO, T., AND VISWANATHAN, H. 2006. Comparison of network coding and non-network coding schemes for multi-hop wireless networks. In *Proceedings of the IEEE International Symposium on Information Theory*.

KATTI, S., KABATI, D., HU, W., RAHUL, H., AND MEDARD, M. 2005. The importance of being opportunistic: Practical network coding for wireless environments. In *Proceedings of the Annual Allerton Conference on Communication Control and Computing.*

KATTI, S., RAHUL, H., HU, W., KATABI, D., MÉDARD, M., AND CROWCROFT, J. 2006. Xors in the air: practical wireless network coding. *SIGCOMM Comp. Comm. Rev. 36,* 4, 243–254.

KEHDI, E. AND LI, B. 2009. Null keys: Limiting malicious attacks via null space properties of network coding. In *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies*.

KROHN, M., FREEDMAN, M., AND MAZIERES, D. 2004. On-the-fly verification of rateless erasure codes for efficient content distribution. In *Proceedings of the IEEE Symposium on Security and Privacy*.

LI, L., RAMJEE, R., BUDDHIKOT, M., AND MILLER, S. 2007. Network coding-based broadcast in mobile ad-hoc networks. In *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies*.

LI, Q., CHIU, D.-M., AND LUI, J. Nov. 2006. On the practical and security issues of batch content distribution via network coding. In *Proceedings of the IEEE International Conference on Network Protocols.*

LIN, Y., LI, B., AND LIANG, B. 2008. Efficient network coded data transmissions in disruption tolerant networks. In *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies*.

LUN, D. S., MÉDARD, M., KOETTER, R., AND EFFROS, M. 2005a. Further results on coding for reliable communication over packet networks. In *Proceedings of the IEEE International Symposium on Information Theory*.

LUN, D. S., RATNAKAR, N., KOETTER, R., EDARD, M. M., AHMED, E., AND LEE, H. 2005b. Achieving minimum cost multicast: A decentralized approach based on network coding. In *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies*.

MÉDARD, M., EFFROS, M., HO, T., AND KARGER, D. R. 2003. On coding for non-multicast networks. In *Proceedings of the Annual Allerton Conference on Communication Control and Computing.*

PARK, J.-S., GERLA, M., LUN, D. S., YI, Y., AND MEDARD, M. 2006. Codecast: A network-coding-based ad hoc multicast protocol. *IEEE Wireless Comm. 13*, 5, 76–81.

PERRIG, A., CANETTI, R., TYGAR, J. D., AND SONG, D. 2002a. The TESLA broadcast authentication protocol. *RSA CryptoBytes 5,* 2, 2–13.

PERRIG, A., SZEWCZYK, R., TYGAR, J. D., WEN, V., AND CULLER, D. E. 2002b. Spins: security protocols for sensor networks. *Wireless Netw. 8,* 5.

RADUNOVIC, B., GKANTSIDIS, C., P. KEY, S. G., HU, W., AND RODRIGUEZ, P. March 2007. Multipath code casting for wireless mesh networks. Tech. rep. MSR-TR-2007-68. Microsoft Research, Redmond, WA.

SUN, K., NING, P., AND WANG, C. 2006a. Secure and resilient clock synchronization in wireless sensor networks. *IEEE J. Select. Areas. Comm. 24,* 2.

SUN, K., NING, P., AND WANG, C. 2006b. Tinysersync: secure and resilient time synchronization in wireless sensor networks. In *Proceedings of the ACM Conference on Computer and Communcations Security.*

TRASKOV, D., RATNAKAR, N., LUN, D. S., KOETTER, R., AND MÉDARD, M. 2006. Network coding for multiple unicasts: An approach based on linear optimization. In *Proceedings of the IEEE International Symposium on Information Theory*.

WANG, D., SILVA, D., AND KSCHISCHANG, F. R. 2007. Constricting the adversary: A broadcast transformation for network coding. In *Proceedings of the Annual Allerton Conference on Communication Control and Computing*.

WIDMER, J. AND BOUDEC, J.-Y. L. 2005. Network coding for efficient communication in extreme networks. In *Proceedings of the ACM SIGCOMM Workshops on Delay-Tolerent Networking*.

WIDMER, J., FRAGOULI, C., AND BOUDEC, J.-Y. L. 2005. Energy-efficient broadcasting in wireless ad-hoc networks. In *Proceedings of the IEEE International Sympossium on Network Coding*.

YU, Z., WEI, Y., RAMKUMAR, B., AND GUAN, Y. 2008. An efficient signature-based scheme for securing network coding against pollution attacks. In *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies*.

ZHAO, F., KALKER, T., MEDARD, M., AND HAN, K. 2007. Signatures for content distribution with network coding. In *Proceedings of the IEEE Internation Symposium on International Theory*.

Received September 2009; revised June 2010; accepted September 2010