Defenses Against Network Coding Pollution Attacks in Wireless Mesh Networks

Irippuge Milinda Perera
ipererа@gc.cuny.edu

May 18, 2011

Abstract

Network Coding is a novel forwarding technique that substantially maximizes the throughput of the wireless mesh networks by requiring the intermediate nodes to mix the input packets before forwarding them. Aside from the maximized throughput, this coding scheme provides lower power consumption, higher reliability and robustness to packet losses due to congestion and line interference. However, this mixing of the packets at the intermediate nodes brings about a severe security threat known as the *Pollution Attack*, in which compromised nodes inject corrupted packets to the network. As a result, a single corrupted packet received by a non-malicious node could contaminate all the output packets generated by that node, and this effect could propagate to other nodes in an epidemic manner.

In recent years, there has been research interest on efficient countermeasures against pollution attacks in network coding for wireless mesh networks. The focus of this paper is to review three of those defense mechanisms. In [8] by Yu et al. an efficient signature-based scheme is proposed which allows the intermediate nodes to detect and filter polluted packets. RIPPLE framework [6] by Li et al. presents a novel homomorphic message authentication code (MAC) as well as a new transmission protocol to overcome this problem. DART [3], proposed by Dong et al., is another framework that attempts to mitigate this problem by using time-based authentication in combination with random linear transformation to mix packets.

This paper first gives an overview of how the network coding works in the wireless mesh network setting and how it brings about the severe security threat, *pollution attack*. Then, the three defense mechanisms [8, 6, 3] are presented. Followed by that is a comparison of the mechanisms, where the overhead on the network throughput introduced by the defense mechanisms is used as the main metric. Finally, the paper ends with a comparison and a conclusion.

Contents

1 Introduction

In traditional networking protocols, a source node sends a data file to a destination node by first breaking it down to a set of data packets and then transmitting those packets one at a time through a set of intermediate nodes [7]. There are two major drawbacks with using this approach in wireless mesh networks. First, when the packets are dropped at the intermediate nodes due to congestion or line interference, the source node has to retransmit them anew. If lots of packet losses occur, the throughput of the network will be severely affected. Second, the packets are sent along a single path although there are potentially multiple paths leading to the destination in a wireless mesh network. This is a waste of network bandwidth.

Network Coding [1] is a new paradigm for forwarding packets that has been shown to maximize the network throughput. It also provides promising solutions to congestion control and reliable packet transfer that the traditional routing protocols fail to address efficiently. The core principle behind this paradigm is that the source node and the intermediate nodes actively mix (or code) the input packets and then forward the mixed packets [4]. The original packets are called *native packets*, while the packets formed after the mixing process are called *coded packets.* Once the destination node acquires enough coded packets, it can decode the native packet generated by the source node. One important feature of network coding is that the loss of a coded packet does not affect the delivery of a native packet generated at the source since a single native packet is coded in multiple coded packets. This makes network coding highly resilient to packet losses at intermediate nodes.

However, the very nature of mixing packets poses new security challenges in network coding. The most disastrous of all such security challenges is the *pollution attack* in which malicious nodes inject corrupted packets into the information flow while pretending to be forwarding packets from the source to the destination. This form of attack is called the pollution attack since the adversarial nodes pollute the information flow by injecting bogus data.

There has been much research interest in designing countermeasures against pollution attack in network coding. These defense mechanisms fall into two broad categories: cryptographic approaches, non-cryptographic approaches. The papers reviewed in this review paper are as follows. [8] by Yu et al. is a cryptographic approach in which the main component is a homomorphic signature scheme. The homomorphism allows the nodes to create valid signatures for new coded packets by using the existing signatures of the mixed coded packets. RIPPLE framework [6] by Li et al. is also another cryptographic approach that makes use of a homomorphic MAC. DART [3] by Dong et al. is a non-cryptographic approach, therefore is not affected by the extra complexity of the previous two approaches introduced by cryptographic operations. Thus, DART is the most computationally efficient of all and its security relies on time asymmetry of the checksums used to validate packets.

Inevitably, when any defense mechanism is used, the size of the coded packets must be increased because of the extra information necessary to detect and filter polluted packets. A good defense mechanism is one that introduces this extra overhead within a reasonable bound (i.e. does not drastically degrade the efficiency). Thus, the main metrics we have used when comparing the different solutions are,

1. The overhead on the network throughput due to the increased size of the coded packets

2. The additional computational overhead introduced by the defense mechanisms on the nodes

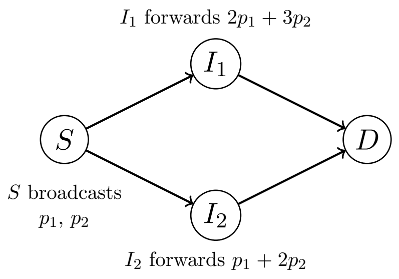
I_1 forwards $2p_1 + 3p_2$
I_1
S
D
S broadcasts p_1, p_2
I_2
I_2 forwards $p_1 + 2p_2$

Figure 1: Nodes I_1, I_2 broadcast a linear combination of the received packets to node D.

2 Network Coding Primer

According to [4], there are two approaches to applying network coding to a wireless mesh network: *intra-flow* network coding and *inter-flow* network coding. Both approaches make use of the broadcast nature and opportunistic overhearing of the nodes in the wireless mesh network in order to achieve the improved performance.

In intra-flow network coding systems, the packets are coded within a single information flow, while in inter-flow network coding the packets are coded within multiple information flows. Although the pollution attack applies in both these approaches, its defenses in inter-flow network coding are still at their infancy due to the increased complexity of inter-flow routing scheme. Therefore, all the defenses considered in this review paper deal with pollution attack in intra-flow setting. Let's concentrate on how intra-flow network coding works.

The idea behind intra-flow network coding in wireless mesh networks is to utilize the opportunistic overhearing with packet coding to achieve the increased throughput. Let's consider the example given in the figure 1 to get a general idea of this process. In this example, the source node S has two packets, p_1 and p_2 to be sent to the destination node D and broadcasts them individually. The intermediate nodes, I_1 and I_2, forward a random linear combination of the packets they receive. The randomness reduces the probability of node D receiving the same linear combination. Once the node D has enough linearly independent coded packets (in this example, 2), it can decode the native packets by solving a set of linear equations.

More specifically, in intra-flow network coding, the packets are delivered in batches known as *generations* [4]. Let G denote a generation of n native packets $p_1, p_2, \ldots, p_n$. A native packet p_i can be viewed as a column vector of m elements in a finite field $\mathbb{F}$ of size q as follows.

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

Then, the generation G becomes,

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

A coded packet contains two components $(\vec{c}, \vec{e})$. $\vec{c} = (c_1, c_2, \ldots, c_n)$, which is called the coding vector, is a random vector in $\mathbb{F}_q$. $\vec{e}$, which is called the coded data, is computed as follows.

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

The source node keeps forwarding newly generated coded packets for the current generation until an acknowledgement message (ACK) is received from the destination node. The ACK is usually sent via a unicast communication channel.

Each intermediate node keeps valid linearly independent coded packets received for the current generation in a buffer and computes and forwards new coded packets by forming random linear combinations of those coded packets in the buffer [4]. For example, let $(\vec{c_1}, \vec{e_1}), (\vec{c_2}, \vec{e_2}), \ldots, (\vec{c_k}, \vec{e_k})$ be the coded packets in the buffer. To compute a new coded packet $(\vec{c'}, \vec{e'})$, the intermediate node picks a random vector $\vec{h} = (h_1, h_2, \ldots, h_k)$ in $\mathbb{F}_q$, and computes,

$$\vec{c'} = \sum_{i=1}^{k} h_i \vec{c_i}$$

$$\vec{e'} = \sum_{i=1}^{k} h_i \vec{e_i}$$

Once the destination node has received n linearly independent coded packets, it can create a system of linear equations containing n equations and n unknowns ($\vec{p_i}$s). Assuming the n coded packets are $(\vec{c_1}, \vec{e_1}), (\vec{c_2}, \vec{e_2}), \ldots, (\vec{c_n}, \vec{e_n})$, each linear equation would be $\vec{e_i} = \sum_{i=1}^{n} c_i \vec{p_i}$. By solving this set of equations the destination node can recover the plain packets $\vec{p_1}, \vec{p_2}, \ldots, \vec{p_n}$.

3 The Problem: Pollution Attack

The wireless mesh networks attract attackers from the inside as well as the outside due to the open nature of the system. Usually, the outside attackers are limited to attacks that do not require access to network resources. These types of attacks include eavesdropping, injection and modification. The inside attackers can mount more powerful attacks since they also have access to the network resources. Therefore, it is usually assumed that the inside attackers have the same capabilities as the legitimate nodes of the system.

Now let's turn our attention to the pollution attack in intra-flow network coding. The pollution attack can be mounted by both outside and inside attackers. The idea here is to inject corrupted coded packets into the information flow. According to the notation introduced in the previous section, a coded packet $(\vec{c}, \vec{e})$ is corrupted if $\vec{e}$ is not a valid linear combination of coded packets created using $\vec{c}$. In other words,

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

Since the intermediate nodes do not have enough information to check the validity of the coded packets, they might incorporate these corrupted packets to generate new coded packets due to the coding scheme employed by them. The danger behind this attack is that a single corrupted packet could potentially contaminate the entire information flow in an epidemic manner within a very small amount of time. As a result, an attacker node could significantly degrade the network performance. Therefore, the key requirement of a good defense mechanism is to allow the legitimate nodes to identify and filter out the corrupted packets generated by the adversarial nodes.

4 Reviewed Defense Mechanisms

As mentioned earlier, the goal of this paper is to review three defense mechanisms against pollution attack in intra-flow network coding for wireless mesh networks. A thorough review of each mechanism is given in the pages that follow.

4.1 An Efficient Signature-based Scheme for Securing Network Coding against Pollution Attacks *by Yu et al.*

The first defense mechanism considered in this paper is an asymmetric-key cryptographic approach proposed by Yu et al [8]. Specifically, it is an efficient signature-based scheme to detect and filter polluted packets. The key idea in this approach is a novel homomorphic signature function.

The digital signature is a mathematical mechanism used to provide the authenticity of digital messages. A traditional signature scheme, which consists of a single signing authority and a set of users, is made up of the following polynomial time algorithms [5].

- *Setup:* The signing authority generates a public, private key pair, keeps the private key to itself and publishes the public key to the users.
- *Signature calculation:* Valid signatures for given messages are generated ONLY by the signing authority using its private key.
- *Message verification:* Any user can verify the validity of the signatures generated by the signing authority using the public key of the signing authority.

The major drawback of this traditional signature scheme is that the valid signatures can only be created by using the private key of the signing authority. Thus, only the signing authority can generate valid signatures since only him possesses the private key. This signature scheme is not suitable for network coding since the mixing process carried out by the intermediate nodes changes the structure of the original messages, thus invalidating the signatures generated by the signing authority.

The homomorphic signature function proposed in this paper allows the intermediate nodes to generated valid signatures for coded packets using the existing signatures of the mixed coded packets [8]. The process is similar to generating new coded packets using the existing coded packets in the buffer of an intermediate node. The key idea behind this homomorphic scheme is to allow the source node to delegate its signing authority to the intermediate nodes. Afterwards, the intermediate nodes can generate the signatures of the newly coded packets without knowing the private key of the source node. At the same time they are not able to generate valid signatures using forged ones. This also prevents the epidemic propagation of polluted packets in the information flow since every node checks the validity of the given messages before employing them in new coded packets.

Basically, this homomorphic signature scheme can be divided to three phases as follows [8]. The second and third stages are run in parallel.

- *Setup:* During this phase, the source node first generates the security parameters (SP), private key (K_{priv}) and public key (K_{pub}). The source node keeps K_{priv} for itself and transmits K_{pub} and SP to the other nodes. SP is used by the intermediate nodes to obtain signature generation capabilities from the source node.

- *Signature calculation:* During this phase, the source generates the signatures of the coded packets using its private key. For a generic coded packet $(\vec{c_i}, \vec{e_i})$, this process can be presented as the follows.

  $$sign(SP, K_{priv}, (\vec{c_i}, \vec{e_i})) = s_i$$

  The intermediate nodes generate the signatures of the newly coded packets using the existing signatures of the mixed coded packets. This process is shown in the following function notation where $(s_1, s_2, \ldots, s_l)$ are the signatures of the coded packets that the intermediate node has used from its buffer to generate the output coded packet $(\vec{c_i}, \vec{e_i})$. One important point to note is that the existing signatures are already verified according to the process given in the message verification phase.

  $$sign(SP, (s_1, s_2, \ldots, s_l), (\vec{c_i}, \vec{e_i})) = s_i$$

  Usually, the signatures are either transmitted individually or attached to the coded packets. Since the signatures are attached to the coded packets in [8], the format of a coded packet becomes,

  $$Coded\ Packet : ((\vec{v}, \vec{e}), s)$$

- *Message verification:* The intermediate nodes as well as the destination node verify the signatures of the received coded packets by using the public key of the source node. The following function captures this process for a generic coded packet $((\vec{v_i}, \vec{e_i}), s_i)$,

  $$verify(SP, K_{pub}, ((\vec{v_i}, \vec{e_i}), s_i)) = \{true, false\}$$

  If the verification succeeds, the coded packets are used for further processing (i.e.: encoding in the case of intermediate nodes, decoding in the case of destination node), otherwise, they are discarded.

  Homomorphic property of the signature function allows batch verification [2] as well. This process vastly reduces the verification overhead when the attack probability is relatively low. For example assume a node receives three coded packets,

  $$((\vec{v_1}, \vec{e_1}), s_1), ((\vec{v_2}, \vec{e_2}), s_2), ((\vec{v_3}, \vec{e_3}), s_3)$$

  The node first generates a new coded packet $((\vec{v_t}, \vec{e_t}), s_t)$ by combining the above three coded packets and verifies its signature s_t. If the verification fails, the node can either check the signatures of the individual coded packets which would take $O(n)$ time or use a binary-search like algorithm which takes $O(log\ n)$.

As a convention of almost all cryptographic protocols, the signature function given in [8] comes with a proof of security. The authors have proven that breaking this signature scheme is equivalent to solving a discrete logarithm problem considered hard to solve in polynomial time. In other words, an attacker node that generates a valid signature for an invalidly coded packet is not only breaking their system but also solving that hard math problem in polynomial time.

The network overhead introduced by this defense mechanism is dominated by the signatures attached to the coded packets. The size of the signature, which is a single group element in $\mathbb{F}_q$, is very minimal compared to the size of the coded packets. However, the signature generation and verification processes impose a significant computational overhead upon the nodes that is inevitable with any asymmetric-key cryptography based approach. This in turn introduces a significant delay to the delivery of the packets.

4.2 RIPPLE Authentication for Network Coding *by Li et al.*

The second defense mechanism revived in this paper, which is codenamed RIPPLE, is a symmetric-key cryptographic approach proposed by Li et al [6]. The RIPPLE framework actually consists of two main components.

- Homomorphic Message Authentication Code (MAC)
- The RIPPLE transmission protocol

The RIPPLE transmission protocol actually exploits the computational nature of the homomorphic MAC to provide better performance. Let's consider these components one at a time.

4.2.1 Homomorphic Message Authentication Code (MAC)

In cryptography, Message Authentication Codes or MACs for short are also used to provide authenticity of digital messages. MACs are the symmetric-key counterparts of the signature schemes which appear in the asymmetric-key setting. Although MACs and digital signatures both serve the same purpose, the way they serve that purpose is quite different. In simple terms, signature schemes are one-to-many (meaning, one generator, many verifiers) while MACs are one-to-one (a sender and a receiver).

A traditional MAC scheme, which is run between a sender and a receiver, consists of three polynomial-time (PPT) algorithms explained below [5],

- *Setup:* This algorithm, which is run by the sender, randomly samples a private key K from a key space $\mathcal{K}$.
- *MAC:* The MAC algorithm, which is also run by the sender, generates the MAC for a given message using the given key. The following function captures this notion.

$$MAC(msg, K) = mac$$

- *Verify:* The verify algorithm is run by the receiver. This algorithm verifies the MAC generated for a message using the same key used to generate that MAC. The function given below captures this notion.

$$\mathit{Verify}(msg, mac, K) = \{true, false\}$$

Using this traditional MAC schemes to provide the authenticity of the coded packets in network coding poses two challenges.

1. Similar to the problem regarding the traditional signature schemes, the mixing process carried out by the intermediate nodes invalidates the MACs of the original packets.
2. Since MAC is a one-to-one scheme (i.e. run between two nodes), each pair of nodes has to establish a private key K before any communication can take place. Establishing K requires the sender to send it though a secure channel, which is another overhead on the system. In order to provide better security, keys must be renewed in a timely fashion. And this key renewal poses more overhead on the system.

The homomorphic MAC scheme introduces in [6] overcomes the first problem by allowing the intermediate nodes to generate valid MACs for output coded packets using the existing MACs of the coded packets used in the mixing process. Specifically, a homomorphic MAC scheme consists of the same three polynomial-time (PPT) algorithms of the traditional MAC schemes plus the **Combine** algorithm shown below, where $\alpha_1, \alpha_2, \ldots, \alpha_d \in \mathbb{F}_q$,

$$Combine((msg_1, mac_1, \alpha_1), (msg_2, mac_2, \alpha_2), \ldots, (msg_d, mac_d, \alpha_d)) = mac_k$$

α value in a given triplet represents how frequently msg of that triplet is used in the coded packet. Thus, mac_k generated by the *combine* algorithm is the MAC of the message,

$$y = \sum_{i=1}^{d} \alpha_i M_i$$

The second problem posed by traditional MAC schemes is inherent in any MAC based system. The RIPPLE transmission protocol dramatically reduces the harm of this problem by exploiting the nature of the homomorphic MAC scheme as explained in the next section.

4.2.2 The RIPPLE transmission protocol

The transmission of packets in this protocol is processed in two coordinates, time and space. According to this protocol, the mesh network is organized in a hierarchical tree structure. The source is the root node, the intermediate nodes are placed at the intermediate levels and the destination nodes are the leaves. A node is called a level-j node if it is j hops away from the source. The protocol allows the maximum number of levels, L, in the wireless mesh network to be a system parameter. The time is divided into uniform segments called the intervals and the packets sent during the interval i are referred to as interval-i packets.

The name "RIPPLE" derives from the fact that the packets are moved in the network from level to level in a wavelike fashion. That is, a batch of packets reaches a level, waits for key disclosure, goes through MAC verification, goes through the coding process, and finally moves to the next level. The following steps informally explain how the RIPPLE transmission protocol works.

1. For a given coded packet *pkt* the source node picks a set of keys $k_1, k_2, \ldots, k_L$ and generates the MACs $mac_1, mac_2, \ldots, mac_L$ using those keys. The source node keeps the keys to itself and forwards *pkt* and its MACs to the level-1 nodes which are its children. Notice that the format of a coded packet is,

$$Coded\ Packet : (pkt, mac_1, mac_2, \ldots, mac_L)$$

2. The level-1 nodes receive the coded packets from the source and saves them in their internal buffers.

3. After a fixed amount of delay, d, the source node discloses k_1 to the level-1 nodes through a secure channel. d is chosen such that all the coded packets have enough time to reach level-1 nodes.

4. After receiving k_1, The level-1 nodes verify the MACs of the coded packets saved in their buffers. Coded packets with valid MACs are kept, others are discarded.

5. Level-1 nodes generate and forward new coded packets to the level-2 nodes. Also note that the MACs of the newly coded packets are also generated using the homomorphic property of the MAC scheme.

The steps 2-5 are repeated at every level until the coded packet reaches the destination nodes. Similar to [8], the destination node sends an ACK message to the source node once it has decoded the original message.

Since the computational complexity of the symmetric-key cryptography is much lower than that of the asymmetric-key cryptography, the computational overhead imposed on the nodes by the RIPPLE framework is much lower compared to the signature based approach given in [8]. However, the additional information attached to a coded packet (L MACs per coded packet) imposes a significant overhead on the network throughput.

4.3 Practical Defenses Against Pollution Attacks in Intra-Flow Network Coding for Wireless Mesh Networks *by Dong et al.*

The third defense mechanism [3] considered in this review paper, codenamed DART, is a non-cryptographic approach proposed by Dong et al. Similar to the RIPPLE scheme, the security of the DART scheme relies on time asymmetry. Legitimacy of the coded packets in the DART scheme is provided by random checksums created by the source node. And the random checksums are digitally signed by the source node in order to provide authenticity.

The DART defense mechanism runs in parallel with the network coding scheme and the following points describe the DART informally.

- Similar to the process of creating and distributing coded packets, the source node *periodically* computes and distributes **random checksum** packets. The format of a checksum packet is,

$$((CHK_s(G), s, t), sig)$$

  where $CHK_s(G)$ is the checksum for the packets in the generation G. s is the random seed used to generate the checksum and t is the time at which the checksum was generated. sig is the signature of the checksum packet created by the source node. A traditional signature scheme is used here because only the source node generates and signs the checksum packets.

- A forwarder node or the destination node maintains two internal buffers, `verified_set` and `unverified_set`, for verified coded packets and unverified coded packets respectively. The intermediate nodes only use the coded packets in the `verified_set` in the mixing process to form new coded packets. Upon receiving a new coded packet, the node places that packet in the `unverified_set` and records the time at which the packet was received.

- Upon receiving a checksum packet, an intermediate node or the destination node first checks if it has already received that checksum by comparing the digital signature of the checksum packet with the signatures of the checksum packets it has saved in its internal buffer. If the checksum is authentic and not a duplicate,

  1. The node forwards the checksum packet to the neighboring nodes.

2. The node uses the checksum to validate any coded packets stored in the `unverified_set` that were received by the node before the checksum packet was created at the source node. The coded packets that pass the verification are moved to the `verified_set` and those that fail the verification are discarded.

   This time asymmetry coupled with the signature attached to the checksum packets prevents the adversarial nodes from creating valid checksum packets for corrupted coded packets.

- Once the destination node has received enough coded packets and decoded the original data packet, it verifies the authenticity of the original data packet with the source node by using an authentication scheme such as MAC or a digital signature scheme.

There are several key ideas to note in the DART scheme [3]. First, the underlying defense mechanism is based on the *checksums*. Compared to the cryptographic mechanisms used in [8] and [6], checksums are very efficient to create and verify since they are based on very cheap mathematical operations.

Second, a node uses a checksum only to verify the packets received by that node before the the checksum was generated. The time at which the checksum was generated is bound to the checksum packets using a digital signature. Because of this time asymmetry and the digital signature, even if an attacker node could generate a valid checksum for a corrupted coded packet, it cannot convince a legitimate node to accept that coded packet.

Third, similar to the batch verification explained in the discussion about [8], DART scheme also allows the nodes to combine several coded packets in the `unverified_set` and verify the packets at once. This further reduces the computational overhead on the nodes when the attack probability is relatively low.

Let's consider how the two metrics we use to compare the defense mechanisms apply in the DART scheme. First, since every coded packet requires a single checksum, the overhead on the network throughput is very low. Second, only the source node creates and signs the checksum packets and creating the checksums is a very cheap operation. The intermediate nodes only verify the checksums. Therefore, the computational overhead imposed on the nodes is very low.

5 Comparison & Conclusion

Network coding is a new routing protocol that increases the reliability and network throughput by exploiting the opportunistic overhearing of the nodes. Due to the very nature of mixing the packets at the intermediate nodes, network coding is prone to the pollution attack. The solutions to the pollution attack in network coding come in two flavors, cryptographic and non-cryptographic solutions.

[8] is based on a special homomorphic signature scheme that appears in asymmetric-key cryptography. Since the operations in asymmetric-key cryptography are very computationally demanding, the computational overhead posed on the nodes by this scheme is high. However, the additional data that the scheme in [8] adds to the existing network coding scheme is constant per packet. Therefore, the overhead on network throughput introduced by [8] is relatively low.

RIPPLE [6] is also a cryptographic solution that is based on a special homomorphic MAC scheme. Since MAC schemes appear in symmetric-key cryptography and the computational complexity of the operations in symmetric-key cryptography is low, the computational overhead on the nodes introduced by RIPPLE is low. But, the additional data that must be added to the existing packets depends on the maximum depth of the network. This poses a high overhead on the network throughput.

DART [3] is a non-cryptographic solution that uses the checksums as the main component to defend against polluted packets. Since checksums are computationally efficient and only one checksum is required for a generation of packets, DART ranks low in both our metrics.

The following table summarizes the results of this review paper. Based on that, DART ranks as the most efficient scheme. RIPPLE and the homomorphic signature based scheme from [8] could be seen as performing equally well. If low computational overhead is a priority, then RIPPLE suits well. And, if the low overhead on network throughput is a priority then [8] becomes a good choice.

	Computational Overhead	Overhead on Throughput
Homomorphic Signature [8]	high	low
RIPPLE [6]	low	high
DART [3]	low	low

References

[1] R. Ahlswede, N. Cai, S.-Y. Li, and R. Yeung. Network information flow. *Information Theory, IEEE Transactions on*, 46(4):1204 –1216, July 2000.

[2] M. Bellare, J. A. Garay, and T. Rabin. Fast batch verification for modular exponentiation and digital signatures. pages 236–250. Springer-Verlag, 1998.

[3] J. Dong, R. Curtmola, and C. Nita-Rotaru. Practical defenses against pollution attacks in intra-flow network coding for wireless mesh networks. In *Proceedings of the second ACM conference on Wireless network security*, WiSec '09, pages 111–122, New York, NY, USA, 2009. ACM.

[4] J. Dong, R. Curtmola, and C. Nita-Rotaru. Secure network coding for wireless mesh networks: Threats, challenges, and directions. *Comput. Commun.*, 32:1790–1801, November 2009.

[5] J. Katz and Y. Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series).* Chapman & Hall/CRC, 2007.

[6] Y. Li, H. Yao, M. Chen, S. Jaggi, and A. Rosen. Ripple authentication for network coding. In *Proceedings of the 29th conference on Information communications*, INFOCOM'10, pages 2258–2266, Piscataway, NJ, USA, 2010. IEEE Press.

[7] L. L. Peterson and B. S. Davie. *Computer Networks: A Systems Approach.* Morgan Kaufmann, 2007.

[8] Z. Yu, Y. Wei, B. Ramkumar, and Y. Guan. An efficient signature-based scheme for securing network coding against pollution attacks. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, 2008.