Opt and Vent: An Efficient Protocol for Byzantine Detection in Wireless Ad Hoc Network Routing

Ioannis Avramopoulos, Hisashi Kobayashi,
Dept. of Electrical Engineering
Princeton University
Princeton, NJ 08544
{iavramop, hisashi}@princeton.edu

Arvind Krishnamurthy
Dept. of Computer Science
Yale University
New Haven, CT 06520
arvind@cs.yale.edu

Randolph Wang,
Dept. of Computer Science
Princeton University
Princeton, NJ 08544
rywang@cs.princeton.edu

Abstract

Deploying multi-hop wireless ad hoc networks in mission critical operations will potentially improve the effectiveness and reduce the cost of these operations. In such environments, the availability of ad hoc networks will be utmost important.

In this paper, we devise mechanisms that can strengthen the resilience of an ad hoc network against attacks, failures, accidents, and selfishness. The adversary is an insider that controls an unknown subset of the nodes or routers. Our objective is to prevent the adversary from interfering with the "healthy" portion of the network. In this direction, we advocate the use of a tool, called Byzantine detection, to identify the adversarial locations at a fine granularity so that these locations can be subsequently bypassed. Our main contribution is a Byzantine detection protocol that achieves a significant reduction in the per-packet computational and communication overhead.

1 Introduction

Multi-hop wireless ad hoc networks have been proposed to, in part, replace fixed infrastructures in areas where the infrastructure has been damaged, is too time consuming or impossible to deploy, or if the (possibly temporary) utility of the infrastructure does not justify its cost. Scenarios of deployment include disaster response and recovery, health care provision, military operations, community networks, and others. Protection of such networks from adversaries can, therefore, be critical.

This paper addresses the problem of securing the routing operation in order to ensure the availability of the packet delivery service in the presence of an *arbitrary* adversary (also sometimes called a *Byzantine* adversary) that controls an unknown subset of the routers (and links). We, thus, assume that all such Byzantine routers can deviate from the employed protocol arbitrarily.

We consider the packet transfer process from source to destination through intermediate hops. We assume that the Byzantine routers will attempt to block the transfer without being identified. Examples of attacks that such routers can mount can be found in [1]. We provide protocols that if employed by a non-faulty source and provided that the destination is non-faulty, ensure either successful packet delivery, or detect the location of the transfer failure (Byzantine detection). These protocols advance prior art by improving the per-packet protection overhead and, thus, incur a large network-wide benefit. As we argue in Section 2, overhead improvements will, in general, come at the expense of increased recovery time. We show that the delay in recovery induced by our performance improvements is minimal.

From a survivability perspective the tools that we provide can reinforce the network's capability to continue to deliver packets in the presence of attacks, failures, and accidents [2]. In future ad hoc networks as, for example, envisioned by the Terminodes project [3], our tools can be used to protect the network against selfishness.

Byzantine detection finds applications in *Byzantine robustness*, which is defined in [4] as the capability of the routing mechanism to deliver packets between source and destination as long as at least one non-faulty path exists between them. By identifying the locations of packet delivery failures and by bypassing these locations, the objective of Byzantine robustness could be achieved. However, the problem of achieving Byzantine robustness in an ad hoc network is harder than the problem of achieving Byzantine detection in such a network. The main reason is the frequent occurrence of innocuous drops because of wireless interference, mobility, and congestion. This topic is further discussed in Section 7.

Byzantine detection can also find applications as a tool against selfishness [3, 5], as a wormhole detection mechanism [6, 7], and in intrusion detection systems [8, 9].

Outline of the Paper: Our primary contribution is the Opt and Vent Byzantine detection protocol, which achieves a significant performance improvement over previous protocols without loss of accuracy in identifying malicious routers. We present the design of this protocol in several steps, which we outline below.

In Section 3, we present a protocol that provides just end-point protection, where the source can detect transmission failures and the destination can authenticate the source and verify the integrity of received data. This protocol does not require intermediate nodes to perform any cryptographic computations, thus aiding fast adoption in ad hoc networks, but it does not necessarily aid the endpoints in detecting malicious routers. This protocol is interesting as it avoids the use of message authentication codes (MAC) for data packets, but instead uses a more efficient mechanism (using hash chains) that provides the same level of protection.

We then consider the Byzantine detection problem (in Section 4) and present a simple protocol that requires the source to send a *Source Authentication Tag* (SAT) [1] along with every message and detect faults by using the feedback mechanism proposed by Awerbuch et al. [10], where the nodes that exhibit delivery failures are queried as to whether they received the correct data packet. The SAT, however, comprises of a series of MACs, one for each router in the path, and therefore imposes a high cryptographic overhead. Furthermore, when a packet is dropped, it is retransmitted with a new SAT, in order to prevent malicious routers from replaying the fault feedback messages, and the new SAT incurs additional overheads in order to generate it at the source and verify it at each intermediate router.

In Section 5, we propose modifications to the Byzantine detection protocol of Section 4 that allow a SAT to be reused in packet retransmissions along with an unprotected *message offset*, a value that is unique to a particular packet retransmission and is reflected back to the source in fault feedback messages. We argue that the protocol cannot be exploited by malicious routers even though the retransmitted packets use the same SAT as the original packet.

Finally, in Section 6, we present our complete design of the Byzantine detection protocol that combines the ideas proposed in Sections 3 and 5. The resulting protocol employs a SAT that does not require the computation of MACs, and it furthermore does not require the recomputation or reverification of the SAT for retransmitted packets.

2 Related Work

Data packet forwarding protection is based on authentication of data and control packets and possibly redundancy across multiple forwarding paths and, thus, comes at the expense of computational and communication overhead. We will describe a whole spectrum of such protection mechanisms with different properties regarding protection overhead and recovery time.

At the left end of the spectrum, there are protocols in which packets and destination ACKs are routed on a single path and are guarded against impersonation and modification using a single message authentication code (MAC) between source and destination. The cryptographic overhead of such protocols is minimal, but they may never find a workable route to the destination even if one exists.

Moving to the right of the spectrum, the single MAC protection per packet is retained but the source makes an effort to deliver its packets by multipath routing. This is the approach taken in the SMT (Secure Message Transmission) protocol that was proposed by Papadimitratos and Haas [11]. Performance measurements show that SMT significantly improves the packet delivery ratio, as well as other performance parameters, under a capable adversary. In this protocol, the adversary is able to decrease the throughput in certain paths, even if none of its routers are positioned in those paths, by mounting DoS attacks on the corresponding intermediate non-faulty routers. SMT, however, monitors the packet delivery ratios and directs traffic to the more reliable paths, thus, increasing the effort that the adversary must put in order to attack all paths.

At the expense of an initialization procedure, the possibility of a DoS attack at the intermediate routers of a path is prevented in the multipath protocol proposed by Perlman [4]. In this protocol the source first establishes paths to the destination using digitally signed control packets whose authenticity is verified at all downstream routers and, subsequently, the authenticator of data packets is not verified at intermediate routers. The adversary cannot prevent the delivery of such packets using routers that are not positioned in the selected paths due to the use of allocated buffers that were reserved in the initialization procedure.

In general, the cryptographic overhead of multipath protection is small but the process of finding a workable route to the destination may take a potentially large amount of time for some sources that depends on the degree of the penetration of the adversary.

Byzantine detection lies at the right end of the spectrum. It was first proposed by Herzberg and Kutten in [12] using an abstract model. More concrete Byzantine detection protocols are proposed in [10] by Awerbuch et al. and in [1] by Avramopoulos et al. Byzantine detection relies on a combination of destination acknowledgements, timeouts (at the source and intermediate routers), fault announcements (FAs), and path-specific authentication in order to pinpoint links to which blame is ascribed for delivery failures. Byzantine detection is the only solution that appears in the literature that could achieve small worst-case recovery times, which we define as the time that elapses from the moment that communication is disrupted until the moment that it is resumed. Byzantine detection comes at the expense of increased computational and communication per-packet overhead.

Awerbuch et al. had the interesting idea of combining the performance advantage of a single MAC protection in the good case that the path is not attacked with the recovery time capability of Byzantine detection when faulty routers are present. In their scheme, data packets are initially protected with a single MAC for the destination and in the event of failures, which are detected by destination ACKs and source timeouts, more MACs are inserted in the packet, for intermediate routers that are called *probes*, using a binary search procedure. The performance advantage of this protocol comes at the expense of increased recovery time (as compared to a protocol where the probe list consists of all downstream routers) since drops will go undetected until the search procedure converges to the link level. We believe that this idea will be important in the future systems that will employ Byzantine detection.

In [1] we proposed the idea of using a source authentication tag (SAT) in a Byzantine detection protocol that does not ensure data integrity at intermediate routers but ensures integrity at the destination, in combination with a mechanism to detect the location of packet forgeries, should the destination determine that the received packet has been forged. The computational advantage that the efficient SAT gives to this protocol comes at the expense of delayed detection in case of a forgery. This idea is similar to the Opt and Vent protocol of this paper. The key differences are that in [1] the SAT consists of MACs rather than the hash elements used in Opt and Vent and, furthermore, that Opt and Vent utilizes a new optimized retransmission procedure which is essential to the performance of a hash-element-based authentication tag.

2.1 Protection Overhead vs. Recovery Time

The above discussion reveals a trade-off between protection overhead and recovery time in data packet forwarding protection mechanisms. Endpoint cryptographic protection with the possible aid of multipath routing has low communication and computational per-packet overhead at the expense of a potentially (exponentially) large recovery delay. Byzantine detection adds both communication and computational per-packet overhead with the benefit of faster recovery. Even within Byzantine detection protocols extra computational overhead can buy recovery time and vice versa.[1] We believe that this trade-off is fundamental, i.e., we do not expect the development of protection mechanisms that will be uniformly optimal in both overhead and recovery time.

2.2 Secure Route Discovery

Secure route discovery protocols such as [14–17] and others are intended to work in conjunction with data packet forwarding protection mechanisms to improve the availability of the routing operation. A discussion of secure route discovery protocols is not within the scope of this paper.

3 Opt and Vent in Endpoint Protection

We begin by considering a cryptographic protection protocol that requires modifications only to the endpoints. It is desirable for such a protocol to satisfy the following requirements:

- The destination should be able to authenticate the source and also guarantee that the integrity of the data is not compromised.
- The source should be able to keep an accurate accounting of transmission failures, so that it can decide when to activate a Byzantine detection protocol or tear down the connection.
- Incurs low cryptographic and communication overheads.

A simple protocol that achieves some of the desired goals works as follows. The source and destination share a secret key, and they append to each data packet and ACK, respectively, a MAC. The MACs enable the source to authenticate ACKs and the destination to authenticate the source and check for data integrity. This is the approach taken, for example, by Papadimitratos and Haas [11].

We will propose a modification to this simple scheme that we call Opt and Vent, which accomplishes the same protection objectives, but does not require the computation of MACs on data packets, thereby lowering the cryptographic overheads. We note that, although the main purpose of this section is to lay the ground for the presentation of the Opt and Vent Byzantine detection

[1] Another example that confirms this ascertainment is the *time optimal* protocol of Herzberg and Kutten that achieves faster detection at the expense of increased communication overhead. Although, as we showed in [13], this protocol has the same worst-case fault detection time with the *communication optimal* protocol, we have more recently observed that this protocol essentially maximizes the worst-case throughput during the recovery period. We defer a detailed description of our observation in the interest of space.

protocol, to the extent of our knowledge, the idea that we present in this section has not been proposed before for endpoint protection.

The idea in Opt and Vent is that the source will *opt* for a more efficient data packet protection mechanism (hash chains) that ensures source authentication (but not integrity), whenever possible. The source will, then, wait for a (one-way) hash of the packet received by the destination, which is protected by a MAC, and will, thus, be able to determine the integrity of the packet at the destination. Should the source determine that the packet received by the destination is not authentic, it will instruct the destination, by means of a MAC-protected packet, to *vent* (or discard) the stored packet. Note that the MACs are used only for ACK and VENT messages and not on the OPT packets.

We will assume an arbitrary adversary that may discard, replay, or modify the data packets and destination acknowledgements, or may inject forged packets with the purpose of forcing the destination to accept forged data, deceive the source that data packets are flowing properly to the destination, or prevent the delivery of legitimate packets. The communication path could also suffer from innocuous drops. These can be drops because of wireless interference, mobility, and congestion. We will also assume *innocuous modifications* to be rare. This assumption is justified by the use of error detecting coding at the wireless links. Should innocuous modifications be common during packet storage and processing at (non-faulty) routers, use of error detecting codes at the internal stages of packet forwarding will prevent these modifications as well.

3.1 Description of the Operation of Opt and Vent

We will assume that the source and destination share a secret key for the cryptographic protection of data packets and ACKs. Data packets at a minimum contain the source and destination IDs (e.g., IPv4 addresses) and a sequence number taken from a monotonically increasing non-wrapping sequence (for replay protection). Destination ACKs at a minimum contain the source and destination IDs and reflect the sequence number of the corresponding data packet.

In our proposed scheme, the source will replace the MAC in certain packets with elements of a hash chain $\mathcal{R}$. $\mathcal{R}$ is precomputed by choosing a random element R_0 and repeatedly applying a one-way hash function h: $R_i = h(R_{i-1}), i = 1, \ldots, k$. If element R_k is securely associated with a source specified sequence number, say K, at the destination, then the source can authenticate sequence number $K+1$ by releasing R_{k-1}. The destination can verify the authenticity of the received ρ by computing $h(\rho)$ and comparing the outcome with R_k. If they are equal, then ρ is authentic. Otherwise, ρ is discarded as a forgery. In general, the source can authenticate sequence number $K+i$ by releasing element R_{k-i}, the authenticity of which is verified by hashing it using h and comparing it with a previously released authentic hash element. Notice that hash elements do not protect the integrity of the corresponding data packet.

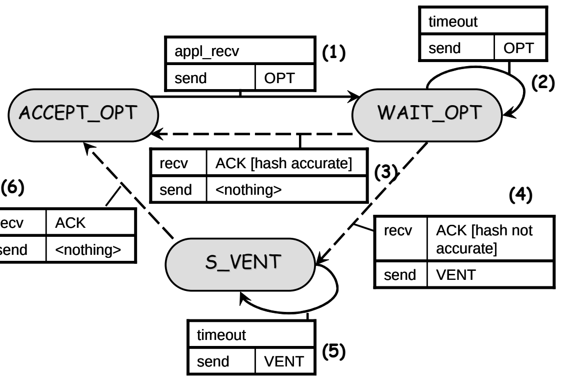

Figure 1: The state transition diagram of a source running the Opt and Vent endpoint protection protocol.

We will view all communication that is being carried out between the source s and destination t as being part of a *connection*. Specifying the layer that this connection should be implemented (e.g., network or transport) is not within the scope of this paper. For clarity of exposition, we will initially describe Opt and Vent using a STOP-AND-WAIT protocol according to which the source waits for an ACK before transmitting the next data packet. The mechanism that allows the source to maintain multiple outstanding packets will be presented subsequently.

3.1.1 Connection Establishment

The source attempts to open the connection by sending a MAC protected packet which associates R_k with sequence number K (and also reserves a sequence number space for use by the connection). In the absence of an ACK the connection establishment packet will be retransmitted unmodified. The destination acknowledges each connection establishment packet by reflecting its sequence number, which is protected by a MAC.

3.1.2 Securing Data Transfer

During the data transfer the source will use two kinds of packets denoted by OPT and VENT. OPT packets are protected with elements of the hash chain, whereas VENT packets are protected with a MAC. Both packets carry data but also have control semantics. The state transition diagrams at the source and destination after the connection establishment phase are shown in Figures 1 and 2.

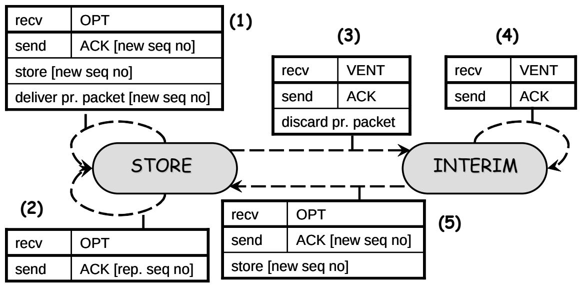

Figure 2: The state transition diagram of a destination running the Opt and Vent endpoint protection protocol.

The source can be in one of three states: ACCEPT_OPT, WAIT_OPT, and S_VENT. In ACCEPT_OPT it is waiting for a packet from the higher layer. When a packet arrives from the higher layer, it will transmit the packet using the OPT option and it will move to WAIT_OPT state (transition labeled 1) and wait for a destination ACK. If a timeout fires in this state, then the OPT packet is retransmitted *unmodified* (2). When it receives the destination ACK, it compares the hash attached in the ACK with the hash of the corresponding OPT packet. If they are equal, the source moves to ACCEPT_OPT state (3). If they are not equal (which implies that the corresponding packet was corrupted), then the source sends a VENT packet and moves to S_VENT state (4). If a timeout fires in this state, then the VENT packet is retransmitted (5). Finally, while the source is in S_VENT state and it receives an ACK it moves to ACCEPT_OPT state (6).

The destination can be in one of two states: STORE and INTERIM. In the STORE state, the destination keeps in memory a received packet whose integrity has not yet been verified. While in STORE state, if the source receives an OPT packet with a new sequence number (with the appropriate hash chain element), the new packet implicitly validates the integrity of the previous packet. The destination will then deliver the stored packet to the higher layer, store the received packet, hash the received packet, attach the hash to the ACK, and transmit a MAC protected ACK (1). If the destination receives an OPT packet with the same sequence number as the stored packet, it will attach the hash of the stored packet to the ACK, transmit the ACK, and discard the received packet (2). (Note that the newly received packet is not stored, nor is its hash used in the ACK.) However, if the destination receives a VENT packet, while in STORE state, it will discard the stored packet, deliver the data in the VENT packet to the higher layer, send an ACK to the source, and move to INTERIM state (3). The ACK should carry an indication that it corresponds to a VENT packet, as the same sequence number is used for both OPT and its corresponding VENT packets.

The purpose of the INTERIM state is to retransmit ACKs of VENT packets (4) so as to achieve synchronization between source and destination after a forgery. The action taken by the destination on receipt of a new OPT packet at INTERIM state is similar to the corresponding action taken by the destination at STORE state with an additional transition to the STORE state (5).

Note that the sequence number is only advanced at the source on receipt of a new packet from the higher (e.g., application) layer (VENT packets carry the sequence number of the corresponding OPT packet) and that retransmitted packets and ACKs are not modified in any way.

3.1.3 Multiple Outstanding Packets

Up to now we described Opt and Vent using a STOP-AND-WAIT protocol. We now address the mechanism by which the source can have multiple outstanding packets in the communication path. Inserting multiple outstanding packets using hash elements from a single chain introduces the complication that the verification of a single hash element (at the destination) may require computing multiple hashes because of possible out-of-order delivery, drops, and retransmission. Furthermore, if packet numbered i is dropped, then releasing packet numbered $i + 1$ before the retransmission of i allows a malicious router to fabricate packet i and instruct the destination to accept the fabricated packet using the hash chain value attached to packet $i + 1$.

One way to address this problem is to use multiple hash chains, where the number of hash chains is equal to the maximum number of outstanding packets w that the source is willing to maintain. The source then authenticates packet numbered i with a value from the hash chain numbered $i\%w$ and uses a stop-and-wait protocol on packets that use elements of the same hash chain.

An alternative approach requires the use of just one hash chain by simply requiring the source to release packet numbered $i + w$ only after receiving ACKs for all packets up to i and confirming that none of those packets have been tampered. If the source receives the ACK for packet i, but it has not received ACK for all packets numbered less than i, it simply waits for the missing ACKs before it releases packet $i + w$. This ensures that malicious routers cannot instruct the destination to accept fabricate packets using the hash chain values associated with later packets.

3.2 Discussion

As we previously mentioned, on receipt of an OPT packet with repeated sequence number, the destination acknowledges the receipt of the first OPT packet carrying the corresponding hash and discards the repeated packet. If the destination discarded, instead, the first packet and stored and acknowledged the most recently received one, then an adversary would gain the advantage to force the destination to accept forged data by performing the following attack: The adversary would first normally forward the first instance of the packet but it would store and drop the ACK. On the second transmission by the source it would forward a forged version of the packet to the destination but replay the previously stored ACK for the source. On its next OPT packet the source would instruct the destination to accept the data of the previous packet as authentic and, thus, the destination would accept the forged data. We will call this the *bolt* attack, and we will see that it has an analog in the Opt and Vent Byzantine detection protocol.

Malicious routers cannot deceive the source into believing that the destination received a dropped packet as, on one hand, the destination ACK is unforgeable and, on the other, it reflects the sequence number of the corresponding data packet and, thus, cannot be replayed. Notice also that by replaying ACKs for packets which are being retransmitted, malicious routers can only signal to the source the, accurate, event of the reception of the corresponding data packet by the destination. Opt and Vent therefore provides accurate accounting of transmission failures at the source, which can decide when to initiate a Byzantine detection procedure or tear down the connection.

We will now explain the utility of the hash chain by considering a hypothetical protocol that operates without the chain (but retains VENT packets) and showing how the protocol can be attacked. In such a protocol, if a malicious router is positioned in the communication path between source and destination, it can, first, completely block the source after connection establishment and, second, continuously mount forgeries to the destination. In the destination, the absence of VENT packets is, therefore, ambiguous: it could be interpreted as either that packets are normally delivered or that the source is completely blocked. It, therefore, seems that, in addition to the negative feedback (communicated by VENT packets), positive feedback (communicated by the hash chain) is also necessary for the correctness of the protocol.

3.3 Performance Comparison

The Opt and Vent protocol imposes the additional memory requirement to the destination, as compared to the standard protocol of MAC protection, that a packet must be stored for an additional round-trip-time upon its reception.

The cryptographic communication overhead of OPT packets is equal to the size of a hash element, while the corresponding overhead of their ACKs is equal to the sum of the sizes of a MAC and a hash. As compared to the standard protocol the ACK overhead is slightly larger than the overhead of standard ACKs. The corresponding overhead in VENT packets and their ACKs is equal to the overhead of standard packets and ACKs. The computational overhead of the standard protocol is the computation of MAC on data packets and ACKs at both the source and the destination. The Opt and Vent protocol, during normal operation, requires the source and the destination to perform a hash chain computation, a hash computation on the data packet, and a MAC on the ACK. This typically results in lower overheads as the MAC on data packets is replaced by a hash computation on data packets and a hash chain computation.

OpenSSL	1500B	256B
Std	26.46 (μsec)	16.69 (μsec)
Opt and Vent	21.89 (μsec)	12.13 (μsec)
cryptlib	1500B	256B
Std	19.38 (μsec)	9.52 (μsec)
Opt and Vent	20.25 (μsec)	10.43 (μsec)

Table 1: Performance comparison between the standard data packet protection mechanism (Std) and Opt and Vent.

In order to measure computational requirements we used two cryptographic libraries: OpenSSL 0.9.7a (http://www.openssl.org/) and cryptlib 3.1 (http://www.cryptlib.orion.co.nz/). Measurements were collected on an Intel Pentium M processor running at 1.5GHz. The cryptographic algorithms used are the HMAC-SHA1 [18] (for MACs) and SHA1 (for hashing data packets). Table 1 shows the computational requirement for OPT packets at the source (or destination) for 1500B and 256B packets assuming that ACKs are 20B without the cryptographic overhead. In the, widely deployed, OpenSSL library Opt and Vent achieves 17.3% decrease in computation for 1500B packets and 27.3% decrease for 256B packets. However, in the cryptlib library Opt and Vent is slower than the standard protocol by 4.5% for 1500B packets and 9.6% for 256B packets.

We also performed measurements using the UHASH and UMAC algorithms (http://www.cs.ucdavis.edu/ rogaway/umac/) and found that with these algorithms Opt and Vent also incurs a small increase in the computational overhead. We are, therefore, investigating the reasons that Opt and Vent performs differently under different cryptographic libraries and MAC algorithms. When applied to Byzantine detection Opt and Vent achieves significant computational savings in all the aforementioned libraries and algorithms.

4 Byzantine Detection Background

In this section, we first define Byzantine detection and then present a Byzantine detection protocol that will serve as a starting point in the development that follows.

4.1 A Definition of Byzantine Detection

Before we define Byzantine detection we will give two other useful definitions:

Definition 1. Given a path in a network that consists of routers and links connecting those routers, we define a *nexus* in the path as a triplet of two adjacent routers and the link that connects them.

Definition 2. We call a packet drop *innocuous* if it is not instigated by the adversary. Examples of innocuous drops can be drops caused by congestion or wireless interference.

We will say that a data packet forwarding mechanism has *Byzantine detection capability*, if it satisfies three properties:

Property 1 (Monitor): The source can monitor the delivery status of its data packets using destination ACKs.

Property 2 (Location): In the event of a failure to deliver a data packet, the source can pinpoint a nexus that will be ascribed with the failure.

Property 3 (Detection): The identification of a culpable nexus by the source implies that at least one of the elements of the nexus innocuously dropped the corresponding packet or its ACK or that at least one of the elements of the nexus is faulty or Byzantine.

We chose to define Byzantine detection at the "granularity" of a nexus for two reasons. The first reason is that the fault detection protocols of Herzberg and Kutten [12], Awerbuch et al. [10], and Avramopoulos et al. [1] detect faults at exactly this level of granularity, and we are not aware of any Byzantine detection mechanism that can operate at a finer granularity. The second reason is that detection at a coarser granularity will, in general, not be sufficient to allow efficient recovery actions at the source.

4.2 A Source Authentication Tag

In [1] we proposed a source authentication tag (SAT) to be used in a Byzantine detection protocol that consists of message authentication codes, one for each downstream router. It uses a structure which ensures the following property: If the tag verifies at one non-faulty router in the path, then it verifies at all non-faulty *downstream* routers in the path, assuming that the source is non-faulty and that the tag is not modified in transit. This property is accomplished by computing each MAC in the tag sequentially from destination to source and by requiring that the MAC for a given router to depend on the MACs of all downstream routers, as illustrated by Figure 3.

We should point out that on successful verification of this tag by a non-faulty router, the router is not guaranteed that all all non-faulty *upstream* routers received the corresponding packet. This SAT is therefore weaker than that proposed by Awerbuch et al. in [10]. In [1] we

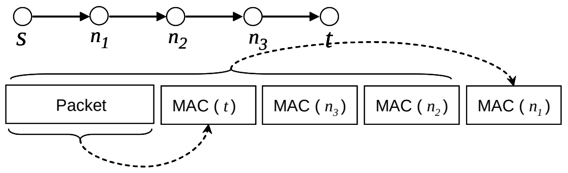

Figure 3: Computation of authentication tags. Source s sends to destination t via the path $\langle s, n_1, n_2, n_3, t\rangle$. Each receiving node needs to verify the authenticity of the packet. The computation of the MAC for n_i receives as input the secret key that s shares with n_i, the packet (or message), and the MACs for $n_{i+1}, \ldots, t$.

argued that this property is not required for the correctness of Byzantine detection and, thus, our more efficient tag is sufficient,[2] but we also suggested an additional protection step of computing MACs between neighboring routers for some networks, including wireless ad hoc ones. David Holmer and Herbert Rubens, in personal communication with the first author of this paper, observed that for wireless ad hoc networks, the additional protection step that we proposed to employ between neighboring routers, may not be necessary. Their suggestion is a topic that we are investigating. We note that this issue does not in any way affect the results that follow.

4.3 A Mechanism for Fault Feedback

In the development that follows the fault feedback mechanism of Awerbuch et al. [10] is essential. In this mechanism, a *probe list* is associated with each message. The probe list of a given path is defined as the subset of routers in the path that participate in Byzantine fault detection. Every node on the probe list is required to send an acknowledgement to the source. Each acknowledgement is protected with a MAC that is computed using the secret key that the source and the probe share and, thus, its authenticity can only be verified at the source. Upon reception of the acknowledgements the source can determine whether the packet reached the destination and, in the event that the packet was dropped, the source can also determine the location of the failure at the granularity of a nexus (by observing the point of disruption of the ACK list). Notice that probes can securely communicate arbitrary feedback to the source by protecting it with the MAC of their ACK. In order to save communication overhead and to prevent the adversary from selectively dropping ACKs, downstream ACKs are accumulated in a single packet before they are forwarded that is encrypted using the secret

[2]This observation is particularly useful when we incorporate Opt and Vent messages into Byzantine detection since using a hash element as the SAT does not ensure the aforementioned property either.

key that the probe shares with the source. Note that the probe list need not contain all the nodes in the path. Instead, the source could refine the probe list based on responses to previous probes, thereby identifying Byzantine nodes using $O(\log n_p)$ probe attempts, where n_p is the number of nodes on path p [10].

We consider the Byzantine detection protocol that combines the SAT of [1] and the fault-feedback mechanism of [10] a starting point in the development that follows.

5 Reducing the Overhead of Retransmissions

In this section, we propose a performance optimization that applies to the retransmission of undelivered packets within the context of a Byzantine detection protocol. On detection of the failure to deliver a packet (which is recognized by the absence of a destination ACK), the source has the options to either retransmit the failed packet or transmit the next packet and let the higher layer decide on an appropriate time to retransmit the failed one as a logically different packet. The performance optimization of this section makes the first scenario a more efficient solution for the source.

5.1 Avoiding sat Recomputation

Note that a straightforward approach to handling immediate retransmissions is to advance the sequence number and, thus, recompute the SAT. The reason for advancing the sequence number is to prevent replay of fault announcements by malicious intermediate routers. Since FAs carry the sequence number of the corresponding data packet, a malicious router could store an FA that was triggered by an innocuous drop in the first transmission and replay it after the retransmission. Since this FA would carry a legitimate sequence number, it would be deemed valid at the source. The source would, thus, associate a fault with a nexus that did not drop the corresponding packet and, therefore, the Byzantine detection property would have been violated.

The protocol that we will propose, on one hand, allows the source to retransmit the failed packet without recomputing the SAT and, on the other, does not have the aforementioned vulnerability regarding the replay of FAs. Our approach is to make FAs differ in different retransmissions by requiring the source to append an *unprotected* offset to the retransmitted packet and the downstream routers to reflect the received offset in their ACKs.

We argue the correctness of this approach by examining the authentication properties of a SAT. We first make a useful observation regarding the extreme case of completely removing the authentication tag: The SAT serves two purposes. The first purpose is to protect against modification of the corresponding packet; the packet that carries an attempted forgery will be dropped at the next downstream router. The second purpose is to allocate a reserved buffer to the intended source; without source authentication, malicious routers can spoof the address of the source in order to cause the source's packets to be dropped at non-faulty routers. If the SAT was removed, then we would create three vulnerabilities in the protocol:

1. The destination would be unsure of the integrity of the received packets and may, thus, accept forged data.

2. The source would fail to detect where its packets were tampered.

3. Malicious routers could cause non-faulty routers to drop legitimate packets (by spoofing the corresponding source).

The first vulnerability does not apply to retransmitted packets because the destination can verify the integrity of the data by the authentication tag. Regarding the second vulnerability, the modification of any part of the packet besides the offset is equivalent to dropping the corresponding packet since the authentication tag will not verify at the next non-faulty downstream router. Furthermore, the location of a modification of the offset can be detected by requiring from downstream routers to reflect their received offset in their ACK. Regarding the third vulnerability, downstream routers cannot verify the authenticity of the offset. However, by forging an offset, malicious routers cannot consume resources reserved for other packets, with different sequence numbers, generated by the same source.

Therefore, the attachment of an unprotected and unpredictable offset to a retransmitted packet is the solution that we adopt for protection against replay of FAs. A random offset, for example, satisfies this requirement. The unpredictability requirement prevents the adversary from manipulating the offset mechanism in order to receive FAs for yet unused offsets. The impact that such an attack would have should the offset be predictable is a topic of future work.

Note that the use of unprotected offsets along with retransmissions gives malicious routers the ability to insert a potentially large number of fake retransmissions and increase the amount of message traffic in the system. We, however, note that these retransmissions could be suppressed by a properly functioning downstream router who had received the earlier transmission, forwarded it, but has not timed-out on the corresponding ACK (or FA, in the case of faults at routers further downstream).

5.2 Avoiding sat Re-verification

In the previous scheme, the computational savings are for the source only. Downstream routers that had already received the packet in a previous transmission must reverify the authentication tag because of the following attack: In a first transmission of a packet a malicious intermediate router would let it pass without modifying it. Assume now that this packet is innocuously dropped at non-faulty downstream link e. In the second transmission the malicious router would modify the packet. The routers upstream to e would let the modified packet pass without verifying its authenticity. On reception of this packet by the router downstream to e, its authenticity would be verified and the packet would be dropped at the non-faulty nexus of e violating Byzantine detection.

We propose now a modification of the above scheme that results in computational and communication savings at downstream routers (in addition to the computational savings at the source). In order to avoid repeated verifications of the authenticity of the retransmitted packet by downstream routers, we require that upon reception of an authentic packet each router keeps it in memory (until reception of an authentic packet from the same source that will instruct its disposal). On subsequent reception of a retransmission of this packet, routers that keep it in memory will forward to the downstream router the version of the packet that is in memory, whose authenticity has already been verified, rather than the received packet, which may have been modified, after setting the (unprotected) offset field to the value of the current retransmission. In this way, each downstream router will verify the authenticity of a packet at most once, irrespective of the number of retransmissions by the source.

Note that this optimization saves not only cryptographic but also communication overhead since each router can tell whether its downstream router has already received the packet from the presence or absence of a cumulative ACK from that router. In the presence of such an ACK for a previous transmission, the only new information that must be transmitted is the current offset.

Requiring downstream routers to keep packets in memory may seem to increase the memory requirements imposed on them. We argue to the contrary due to the use of *reserved buffers* as suggested in [1]. By using reserved buffers, the total number of congestion drops is contained in an ad hoc network to congestion drops due to contention of the wireless medium. Without buffer reservations malicious routers could cause network-wide disruptions by overwhelming the network with their own packets and, thus, resources would be denied to the packets of non-faulty sources.

6 The Opt and Vent Byzantine Detection Protocol

In this section we will use the Opt and Vent idea to construct a Byzantine detection protocol on the basis of the protocol of Section 4.

6.1 A First Step

We modify the Byzantine detection protocol that is obtained by combining the use of SAT of [1] and the fault-feedback mechanism of [10]. In a straightforward application of Opt and Vent to Byzantine detection, the SAT would be replaced with a single hash element and each downstream router would append a hash of the received packet to its ACK. By receiving the hashes the source would be able to detect whether the packet had been modified and, if so, would also be able to pinpoint the location of the modification. This is the protocol that we will investigate in this section. In the next section, we will propose a variation of this protocol that improves performance.

The replacement of the SAT with elements of a hash chain requires addressing the following challenge: In the event that a packet is dropped, its corresponding hash element will not become available at all downstream routers. If the source transmitted the next packet (which could be the lost packet or a new packet in the data stream) with the next hash element, then some downstream routers would need to compute two hashes to verify its authenticity, which is not desirable for performance reasons. In order to gain the computational advantage of the hash chain, dropped packets must be retransmitted without changing the hash element. However, in the event that the retransmitted packet that carries the reused hash element is dropped, the source must still be able to detect the location of the delivery failure.

Note that the Opt and Vent endpoint protection protocol had weaker requirements with regards to analyzing transmission failures. It merely required the source to identify that a packet was lost; it does not require the source to pinpoint the location of the failure. It can therefore reuse hash elements for retransmitted packets and also let the acknowledgements to retransmitted packets be identical. (The destination acknowledges receipt of the first packet that carries a given sequence number by including its hash value in the ACK.) This feature does not prevent the source from detecting transmission failures because a possible replay of the retransmitted packet's ACK by the adversary signals to the source the accurate event of the reception of the first packet by the destination. However, the Byzantine detection protocol uses negative feedback (in the form of fault announcements) in addition to ACKs, and these FAs require stronger protection, because malicious routers can persistently replay FAs and, inaccurately, associate faults with a link that failed only once. Thus, FAs for retransmitted packets should differ.

We address this challenge by making use of the retransmission procedure of Section 5 that associates a random unprotected offset with the retransmitted packet. The difference with Section 5 is that the SAT is replaced with hash elements and that downstream routers include in ACKs a hash of the received packets. This difference does not in any way affect both the offset mechanism and the optimization regarding avoiding repeated verifications of retransmitted packets. In fact, the latter method, furthermore, effectively counters the *bolt* attack of Section 3 that has a direct analog in the Byzantine detection protocol.

6.2 A Performance Optimization

As a final step, notice that the data transmitted by the source is intended to be consumed at the destination alone. In the previous section, the hashes that the source receives from intermediate routers serve the sole purpose of detecting the *location* where the packet was modified. Without these hashes, the source is still able to tell whether the data was received by the destination and whether the received data is untampered. We now design a protocol that does not require intermediate routers to perform a hash on the received packets. In the event though of the detection of a modification, from the comparison of the source and the destination hash, the protocol must be able to offer the source the capability to pinpoint the location of the modification.

The idea of a mechanism that offers such capability is the following: If we require from intermediate routers to store received packets until the next authentic hash element arrives, then on detection that the destination received a forged packet, the source can query intermediate routers for hashes of their received packets. By comparing these hashes with the hash of the original packet, the source can, thus, determine the location of the modification of the packet.

Realizing this idea requires an appropriate handling of retransmissions. Since failed packets are retransmitted without a new SAT, which version of the packet should intermediate routers store? The retransmission mechanism of Section 5, according to which routers keep in memory the first received packet with an authentic tag and retransmit this packet on request from the source, suffices. The reason is that once a malicious router modifies a packet, state of the modification will be kept at the first non-faulty downstream router (and beyond) and, furthermore, this state cannot be later nullified by the malicious router.

It is informative to show how the protocol would fail if downstream routers kept in memory the first received packet with a new authentic tag but retransmitted the currently received packet instead of the one that is stored in memory. A malicious intermediate router would let the first packet sent by the source pass unmodified. Lets assume that this packet got dropped at non-faulty link e. When the retransmitted packet would arrive at the malicious router, it would modify it and forward the modified packet. Upon reception of the destination ACK the source would detect the modification and send a query packet. The answer to the query of the router upstream to e is the hash of the original packet whereas the answer of the router downstream to e is the hash of the modified packet. The source would then account a modification to the non-faulty nexus of e.

6.3 Overview of Opt and Vent

The Opt and Vent Byzantine detection protocol operates in three phases. The purpose of the first phase is to bootstrap the hash chain whose elements will be subsequently used for the SAT of data packets. In this phase, a second hash chain is also set up whose elements will be used as the SAT of the query packets, which are dispatched in the event of the detection of a modification. The SAT used in the first phase is per Section 4 and retransmissions are handled per Section 5. On completion of the first phase, we follow the protocol of Section 6.2 to transmit data. Packets transmitted during this phase are called OPT *packets* (as in Section 3). On detection of a modification of an OPT packet, a VENT-QUERY *packet* is dispatched to query downstream routers of hashes of their received packets in order to pinpoint the location of the modification and also inform the destination to ignore the corrupted packet. VENT-QUERY packets are handled per Section 6.1.

After the detection of the nexus that modified the packet, the action taken by the source can be the deletion of the corresponding link from its topological map (assuming that modifications are rarely innocuous), if the nexus corresponds to a physical link, or an increase in the number of probes, if the link in the nexus corresponds to a path in the physical topology. Note that, in order to avoid detection, malicious routers can persistently drop VENT-QUERY packets, which will induce, however, a large penalty on their corresponding incident links, thereby resulting in their eventual deletion from the topological map.

6.4 Performance Comparison

The cryptographic communication overhead of OPT packets and their ACKs is the hash element (that serves as the SAT), the MACs in the cumulative ACK, and a destination hash. As compared to the protocol of Section 4 the overhead of the SAT is reduced m times, where m is the number of downstream routers, and the overhead of the cumulative ACK is increased approximately

OpenSSL	two-hop	ten-hop
Std (μsec)	107.69	634.54
OPT (μsec)	64.03	309.30
VENT (μsec)	77.14	469.02
cryptlib	two-hop	ten-hop
Std (μsec)	83.66	551.68
OPT (μsec)	59.05	311.05
VENT (μsec)	73.07	491.59

Table 2: Comparison of the computational overhead of Opt and Vent packets.

OpenSSL	total	src	interm	dest
Std (μsec)	634.54	317.27	290.87	26.42
OPT (μsec)	309.30	148.47	138.97	21.88
VENT (μsec)	469.02	169.28	277.82	21.88
cryptlib	total	src	interm	dest
Std (μsec)	551.68	275.84	265.04	19.32
OPT (μsec)	311.05	145.14	145.42	20.15
VENT (μsec)	491.59	172.11	298.30	20.17

Table 3: Breakdown of the computational overhead of Opt and Vent packets for a path of ten hops.

by one hash. The exact change depends on the encryption algorithm; assuming that the algorithm is AES in CBC mode, then, as the ACKs are accumulated, padding is necessary to ensure that the encrypted quantity is a multiple of the block size. In VENT packets the reduction of the size of the authentication tag is counterbalanced by the hashes that intermediate routers append in their ACKs.

We measured computational requirements using the OpenSSL and cryptlib libraries, an Intel Pentium M processor running at 1.5GHz, and the HMAC-SHA1 and SHA1 algorithms, as in Section 3. Table 2 shows the total cryptographic computational overhead incurred by all the nodes in the path, which includes computations for data packets as well as their ACKs, assuming that the size of data packets is 1500B.

The total computational overhead for OPT packets is significantly less than that of the protocol messages of the Std protocol (of Section 4). The savings vary from 29.4% (for a two-hop path using the cryptlib library) to 51.3% (for a ten-hop path using the OpenSSL library). The computational savings for the VENT packets is a bit smaller since it requires hashes from all downstream routers, and they vary from 10.9% (for a ten-hop path using cryptlib) to 28.4% (for a two-hop path using OpenSSL). Note however that VENT packets are dispatched only after the detection of a forgery, and the protocol is therefore likely to enjoy the performance savings of OPT packets for the common case. In fact, the performance savings would be even higher when the system experiences innocuous drops, as the Opt and Vent protocol employs the efficient retransmission protocol of Section 6.1, while the Std protocol would incur repeated overheads for generating and verifying the SAT.

Table 3 provides the breakdown of the total processing overhead. It presents the overhead experienced by the source (src), the total overhead incurred by the nine intermediate routers (interm), and the destination overhead (dest). The processing overhead of OPT packets at the destination and of VENT packets at all downstream routers may decrease or increase (as compared to Std packets) depending on the library. However, the *total* overhead of OPT packets is less in both libraries because of the significant savings at both the source and the intermediate routers. Similarly, the total overhead of VENT packets is less in both libraries because of the significant savings at the source.

It is informative to compare the performance of Opt and Vent with the basic protocol of Section 4 in a scenario where a malicious intermediate router decides to create a disruption in the communication of the corresponding path. Under the protocol of Section 4, the strategy that the adversary should follow is to drop packets. Under Opt and Vent, because of the efficient handling of retransmissions and the good performance of OPT packets, it is better for the adversary to first modify a packet and then drop the VENT packets that follow. Thus Opt and Vent incurs a delay in recovery of *one transmission*, which is minimal.

7 Future Work

In Section 1 we mentioned that innocuous drops complicate the problem of achieving Byzantine robustness, despite the existence of efficient Byzantine detection algorithms. The most challenging type of innocuous drops to cope with are congestion ones. The reason is that if routing decisions depend on congestion (and, thus, delay), then the routing mechanism may become unstable (in the *absence* of an adversary) [19]. Notice that the occurence of congestion drops is unavoidable in ad hoc networks because of the contention-based wireless channel. In the absence of congestion drops, a simple path calculation mechanism that penalized each link with the $-log$ of its success probability could have been sufficient for routing purposes.

Furthermore, driven by our observation of the tradeoff between protection overhead and recovery time in data packet forwarding protection mechanisms, we are investigating algorithms that will balance the protection overhead and the recovery time capability of the protection mechanism based on an estimate of the threat that the adversary poses to the packet delivery service.

Acknowledgements

We would like to thank Qiang Huang for asking a question that stimulated our thinking.

References

[1] I. Avramopoulos, H. Kobayashi, R. Wang, and A. Krishnamurthy, "Highly secure and efficient routing," in *Proc. IEEE Infocom 2004*, Hong Kong, Mar. 2004.

[2] R. Ellison et al., "Survivability: Protecting your critical systems," *IEEE Internet Computing*, pp. 55–63, Nov.-Dec. 1999.

[3] J.-P. Hubaux, T. Gross, J.-Y. Le Boudec, and M. Vetterli, "Towards self-organized mobile ad hoc networks: The terminodes project," *IEEE Communications Magazine*, pp. 118–124, Jan. 2001.

[4] R. Perlman, *Network Layer Protocols with Byzantine Robustness*, Ph.D. thesis, Massachusetts Institute of Technology, Aug. 1988.

[5] L. Buttyan and J.-P. Hubaux, "Stimulating cooperation in self-organizing mobile ad hoc networks," *ACM/Kluwer Mobile Networks and Applications*, vol. 8, no. 5, pp. 579–592, 2003.

[6] Y.-C. Hu, A. Perrig, and D. Johnson, "Rushing attacks and defense in wireless ad hoc network routing protocols," in *Proc. 2003 ACM Workshop on Wireless Security*, San Diego, CA, Sept. 2003.

[7] Y.-C. Hu, A. Perrig, and D. Johnson, "Packet leashes: A defense against wormhole attacks in wireless networks," in *Proc. IEEE Infocom 2003*, San Fransisco, CA, Mar. 2003.

[8] A. Mishra, K. Nadkarni, and A. Patcha, "Intrusion detection in wireless ad hoc networks," *IEEE Wireless Communications Magazine*, pp. 48–60, Feb. 2004.

[9] Y. Zhang, W. Lee, and Y.-A. Huang, "Intrusion detection techniques for mobile wireless networks," *ACM/Kluwer Wireless Networks*, vol. 9, no. 5, pp. 545–556, 2003.

[10] B. Awerbuch, D. Holmer, C. Nita-Rotaru, and H. Rubens, "An on-demand secure routing protocol resilient to byzantine failures," in *Proc. 2002 ACM Workshop on Wireless Security*, Atlanta, GA, Sept. 2002.

[11] P. Papadimitratos and Z. Haas, "Secure data transmission in mobile ad hoc networks," in *Proc. 2003 ACM Workshop on Wireless Security*, San Diego, CA, Sept. 2003.

[12] A. Herzberg and S. Kutten, "Early detection of message forwarding faults," *SIAM J. Comput.*, vol. 30, no. 4, pp. 1169–1196, 2000.

[13] I. Avramopoulos, H. Kobayashi, and R. Wang, "A routing protocol with byzantine robustness," in *Proc. 2003 IEEE Sarnoff Symposium*, Princeton, NJ, Mar. 2003.

[14] Y.-C. Hu, A. Perrig, and D. Johnson, "Efficient security mechanisms for routing protocols," in *Proc. Network and Distributed System Security Symposium, NDSS '03*, San Diego, CA, Feb. 2003.

[15] Y.-C. Hu, A. Perrig, and D. Johnson, "Ariadne: A secure on-demand routing protocol for ad hoc networks," in *Proc. 8th Annual International Conference on Mobile Computing and Networking*, Atlanta, GA, Sept. 2002.

[16] K. Sanzgiri, B. Dahill, B. Levine, C. Shields, and E. Belding-Royer, "A secure routing protocol for ad hoc networks," in *Proc. 2002 IEEE International Conference on Network Protocols (ICNP)*, Paris, France, Nov. 2002.

[17] P. Papadimitratos and Z. Haas, "Secure routing for mobile ad hoc networks," in *Proc. Communication Networks and Distributed Systems Modeling and Simulation Conference*, San Antonio, TX, Jan. 2002.

[18] H. Krawczyk, M. Bellare, and R. Canetti, "Hmac: Keyed-hashing for message authentication," in *Internet RFC 2104*, Feb. 1997.

[19] J. Wang, L. Li, S. Low, and J. Doyle, "Can shortest path routing and tcp maximize utility," in *Proc. IEEE Infocom 2003*, San Frascisco, CA, Mar. 2003.