Nicephorus: Striking a Balance between the Recovery Capability and the Overhead of Byzantine Detection

Ioannis Avramopoulos * Arvind Krishnamurthy † Hisashi Kobayashi* Randolph Wang ‡

Abstract

Perimeter security mechanisms, such as firewalls, do not provide adequate protection once the adversary has penetrated the perimeter either because the firewall was bypassed or because the adversary is an insider trusted party. By gaining presence inside a communication network, even at a few locations (routers and links) of strategic importance, the adversary gains the advantage to disrupt the operation of the whole network, in the absence of suitable protection mechanisms.

In this paper, we contribute to protocols (that are termed Byzantine detection protocols) that enable a network to identify the (initially unknown) locations of the adversary inside the network, so that they can be subsequently bypassed, by improving their capability for recovery as well as their overhead. The fault detection state that is obtained by Byzantine detection pertains to triplets of a link and its two adjacent routers without any implication on the faultiness of individual elements of the triplet. Our first contribution is an algorithm that estimates the degree of penetration of the adversary inside the network given this ambiguous fault detection state and its application to the problem of balancing the performance of the detection protocol with its recovery time capability (two parameters that, as we show, can be at a trade-off). Our second and third contributions are on the Byzantine detection mechanisms. We present a generic mechanism that a source router can use to securely obtain arbitrary router-specific feedback and a data forwarding protocol with Byzantine detection capability that significantly improves the performance of previous proposals.

1 Introduction

Networks in operation and their network protocols have adequate, although by no means perfect, resilience to fail-stop failures. However, they enjoy little to no protection against arbitrary (Byzantine) failures that may occur because of human error (software bugs and misconfigurations) or malicious attacks by an adversary. Byzantine failures can severely affect the availability of the network service.

One way to protect networks against attackers is by a two-layer approach. In the first layer, a firewall [5] is deployed that makes it hard for malicious outsiders to hack into the network's hosts and routers. In the second layer, an Intrusion Detection System (IDS) is deployed at key places in the network that statistically analyzes snooped traffic and raises alarms when intrusions are detected. This approach that does not take the dispositions of the security of routing into account is not well-suited for detecting compromised routers because, first, the IDS itself may be compromised, second, compromised routers can avoid the traffic collection points because of the rich connectivity of the network, and, third, compromised routers can masquerade as other (non-faulty) routers while generating traffic that passes through the collection points. We note, however, that IDSs which are particularly designed to maintain the network's availability in the presence of Byzantine failures have been recently proposed in [16, 17].

A second approach to protecting networks from insider attacks is to secure topology or route discovery. Several protocols have been proposed in the literature in this regard [8, 10–13, 18, 20, 24]. However, even if topology discovery is secured, the adversary may not be deprived of its capability to, in part, prevent discovery of non-faulty routes, deprave discovered routes with non-existent links, or position its routers in discovered routes with the purpose of attracting data traffic in order to block it. We, therefore, argue that the protection of data packet forwarding is of equal importance to the protection of topology discovery.

Byzantine detection [3, 4, 9] is a promising approach to deal with insider attacks mounted from initially trusted routers. It uses a combination of destination acknowledgements, timeouts, fault announcements, and appropriate authentication mechanisms so that either the normal flow of packets is not obstructed or that the

*Dept. of Electrical Engineering, Princeton University. Email: {iavramop, hisashi}@princeton.edu
†Dept. of Computer Science, Yale University. Email: arvind@cs.yale.edu
‡Dept. of Computer Science, Princeton University. Email: rywang@cs.princeton.edu

faulty or adversarial locations are identified at the granularity of a "link". Byzantine detection is the approach that we adopt in this paper in order to counter the aforementioned malicious routers that exhibit arbitrary behavior.

In this paper, we make three distinct contributions. First, we attempt to dynamically balance the overhead of Byzantine detection and the recovery capability (i.e., the capability to converge to non-faulty paths after the adversary has disrupted communication). In this regard, we begin by noting that a previously proposed Byzantine detection protocol can provide improved recovery capability as compared to a more basic form of Byzantine detection, at the expense, however, of additional overhead. We, therefore, propose to utilize the former protocol under an adversary with a large degree of penetration and the latter protocol when the threat that the adversary poses is low. However, the degree of penetration of the adversary in the network is not readily available from the state that Byzantine detection offers, which contains an ambiguity on the faultiness of individual routers and links. We, thus, propose a mechanism that can estimate the threat that the adversary poses to the packet delivery service and can switch between the two types of protocols based on this estimate.

Second, we design a mechanism that could be used by a source router to query all of the routers along a path for some router-specific information in a secure and efficient manner. The source router can use this mechanism to gather arbitrary feedback from each one of the routers, while being assured that the query mechanism by itself has the Byzantine detection properties of being able to identify misbehaving elements.

Finally, we propose an efficient data forwarding mechanism that exhibits Byzantine detection properties and uses the secure query mechanism for bootstrapping purposes. Our resulting protocol is more efficient than what we had proposed in [3] and the efficiency is achieved without loss of recovery capability.

In our work, we do not place any restrictions on the attacks that the adversary may mount from the routers and links that it controls. Consequently, our approach is relevant to the various kinds of Byzantine failures that occur in operational fixed infrastructure networks [15, 23]. Such failures can have adverse impacts and also be time-consuming to fix, and tools that provide resilience to such failures are, therefore, highly desirable. The tools that we provide in this paper are also applicable to other types of networks, namely, overlay and wireless ad hoc. Byzantine detection has the potential to provide recovery in highly adverse environments containing multiple compromised routers that maximize their impact by, for example, announcing false connectivity information so as to increase the volume of traffic that they receive and subsequently drop the received traffic.

2 Byzantine Detection Background

In this section we give background information on Byzantine detection that is required in the sections up to Section 5. At Section 5 we give additional background information on Byzantine detection that is required in subsequent sections.

2.1 A Definition of Byzantine Detection

Before we define Byzantine detection we give two other useful definitions. Given a network that consists of routers and links connecting those routers, we define a *nexus* of the network as a triplet of two adjacent routers and the link that connects them. We also call a packet drop *innocuous*, if it is not instigated by the adversary. Wireless interference and congestion are reasons for innocuous drops.

We say that a data packet forwarding mechanism has *Byzantine detection capability*, if a source can monitor the status of delivery of its data packets using destination ACKs and in the event of a packet delivery failure the source can pinpoint a nexus that will be ascribed with the failure. This nexus must also satisfy the property that at least one of its elements innocuously dropped the corresponding packet or its ACK or that at least one of the elements of the nexus is faulty or Byzantine. A basic form of Byzantine detection achieves this by a combination of destination acknowledgements, timeouts (that are set at the source and intermediate routers), fault announcements (FAs) (that are released and propagated toward the source on timeout expiration by intermediate routers), and appropriate authentication mechanisms [3, 9].

Note that if a nexus has been detected as culpable by Byzantine detection, there is no implication on the culpability of the individual components of the nexus. We are not aware of any fault detection mechanism that can, in general, ascribe failures to individuals routers or communication links given the presence of malicious routers in the network. Thus, fault detection state obtained by Byzantine detection entails ambiguity regarding faultiness of individual routers and links.

2.2 Balancing Performance and Recovery Time

Byzantine detection mitigates the damage that the adversary can inflict to source-destination pairs that are connected by a non-faulty path; without any mechanism to identify and bypass locations controlled by the adversary, even such pairs could be deprived of their ability to communicate. However, Byzantine detection does not abate any damage that the adversary can inflict to those pairs. In particular, the adversary can try to decelerate the convergence to non-faulty paths so as to prolong the service disruption. As a defense mechanism against such attacks we will first present a protocol that offers faster recovery capability at the expense of additional per-packet overhead. We will then propose a rule for deciding when this protocol should be enabled.

2.3 On the Time-Optimal Protocol of Herzberg and Kutten

In [9] Herzberg and Kutten presented a Byzantine detection protocol in which *every* downstream router is required to acknowledge each packet upon reception (compare this with the single destination ACK of the basic form of Byzantine detection of Section 2.1) and in which the source and intermediate routers set multiple timeouts to receive the ACKs of all downstream routers. If any timeout fires, then an FA is propagated upstream. This protocol was termed *time-optimal* as the progress of the packet is continuously monitored and, therefore, if the packet is dropped at a router, then the ACKs of its downstream routers will not follow. In this paper, we utilize this time-optimal protocol in order to improve the worst-case recovery time of Nicephorus.

In this regard, we observe that an adversarial router that normally forwards data packets but drops destination ACKs can delay the detection of the corresponding drop by the source. This strategy would seem plausible in order to delay recovery; however, by adopting this strategy, the adversary does not obstruct the flow of packets to the destination. Assuming that the source will continue its effort to find a workable route to the destination, on successful discovery of such route, it will be able to query the destination on which packets were successfully delivered and, thus, it will not have to retransmit those packets. The time-optimal protocol of Herzberg and Kutten, therefore, seems to be maximizing the worst-case throughput during the recovery period.

3 A Metric of the Threat of the Adversary

In this section, we propose a decision rule for enabling the aforementioned time-optimal Byzantine detection protocol. We base the decision on an estimate of the number of faulty network elements (i.e., routers and communication links). If the estimate is low, then the protocol that is outlined in Section 2.1 will be preferable, whereas, if the estimate is high, then the time-optimal protocol of Section 2.3 will be preferable. Specifying an exact threshold of the decision rule is not within the scope of this paper.

Without a-priori information on the faultiness of network elements, the fault detection state, i.e., the collection of detected faulty nexuses, is the only information to derive such an estimate from. The following three choices seem natural to use as the estimate: a) the maximum number of elements that explain the fault detection state, b) the number of detected faulty nexuses, or c) the minimum number of elements that explain the fault detection state.

However, the number of faulty nexuses can be considerably larger than the number of faulty routers, since if a router is faulty, then so are all of its incident nexuses. Thus, the first two approaches would tend to exaggerate the adversary's potential to inflict damage. Furthermore, if fault detection state is shared among sources, the first two choices would give even to an adversary that controls a few routers the capability to coerce a high overhead protection mechanism by spuriously announcing a large number of nexuses as faulty.

We, therefore, adopt the third measure and estimate the threat that the adversary poses by the minimum number of elements that explain the fault detection state, which is obtained at any given router by including the nexuses that this router has detected as faulty as well as the nexuses that other routers have announced as faulty. Note that this measure does not give to adversarial routers the opportunity to manipulate their ability to announce non-faulty nexuses as faulty so that the higher overhead protection mechanism would be activated even if the threat is low. We will show in the next section that this minimum number of elements is equal to the minimum number of routers that explain the fault detection state assuming that communication links are not faulty.

The merits of estimating the number of faulty elements by the minimum number of routers that explains the fault detection state for the purposes of enabling the time-optimal protocol are the following:

1. The minimum number of routers that explains the fault detection state is a lower bound on the number of faulty elements in the network. An adversary that has already compromised a considerable number

of routers is a capable adversary that has the potential to increase the degree of its penetration. Under such a sophisticated attack, non-faulty routers would prefer not to severely delay the establishment of communication, at the expense of lower performance, from a high quality connection that would need a large amount of time to be established.

2. Byzantine detection provides fault detection state one nexus at a time and, therefore, the fault detection state will most of the time be partial (it will be complete only if all the nexuses of faulty routers have been detected as faulty). An estimate of the faulty routers can, therefore, be used to project the potential future damage.

3. Since faulty routers can misbehave selectively, by sharing fault detection states among different sources and estimating the number of faulty routers from the cumulative state, sources can project potential future damage. Note that this estimate protects non-faulty routers from spurious announcements made by faulty routers that have the purpose of forcing non-faulty routers to adopt higher overhead protection mechanisms under an attack with low impact. For example, if a faulty router announces a large number of non-faulty nexuses as faulty, then with our estimation procedure it is more likely that the faulty router will be estimated as faulty rather than the non-faulty routers in the spuriously announced nexuses.

4 Obtaining Bounds on the Number of Faulty Elements

The fault knowledge that is obtained by Byzantine detection contains a fundamental ambiguity: The detection of a faulty nexus implies that the corresponding upstream router, or link, or downstream router is faulty without any implication about the faultiness of a specific element of the nexus. Thus, a given set of detected faulty nexuses can be *explained* by many possible *faulty configurations*; in other words, a given set of faulty routers and links can give rise to many possible collections of detected faulty nexuses.

Ideally, we would like to be able to identify precisely which elements are faulty in a faulty configuration. Using currently known techniques this is impossible in the presence of Byzantine elements. Our approach in this section is to *estimate the number* of faulty elements in a faulty configuration using the observations obtained from Byzantine detection. More specifically, we present algorithms for obtaining lower bounds on the number of faulty elements in the network. This information can be useful in estimating the degree of penetration of the adversary and, thus, balancing performance with recovery time as pointed out in the previous section.

4.1 Definitions and Notation

We represent a communication network with a graph $G(V, E)$, where V is the set of vertices (routers) and E is the set of edges (links). We assume that the edges of the graph are undirected. We define a *faulty element configuration* f of G to be a function $f : V \cup E \to \{0, 1\}$. If $f(\epsilon) = 1$ then ϵ is assumed faulty and non-faulty otherwise. We similarly define a *faulty router configuration* g of G to be a function $g : V \to \{0, 1\}$. We define the degree β_f of a faulty element configuration f by the following sum: $\beta_f \equiv \sum_{\epsilon \in V \cup E} f(\epsilon)$. The degree β_g of a faulty router configuration g is defined similarly.

We denote a nexus identified by source s as $s : u - v \equiv \{s, u, (u, v), v\}$. We call the collection Φ_s of detected faulty nexuses by a source s the *local fault detection state* of that source: $\Phi_s = \{s : u - v |$ nexus $u - v$ is detected as faulty by $s\}$.

We allow sharing of the individual Φ collections between different sources. Assume that source s has received the local fault detection states from a subset S of the sources in the network. We call the collective information at s that results from this sharing the *extended fault detection state* Φ_s^e of s: $\Phi_s^e = \Phi_s \cup \Phi_S$, where $\Phi_S = \cup_{\sigma \in S} \Phi_\sigma$.

We say that a faulty element configuration f *explains* the local fault detection state Φ_s of source s, if $s : u - v \in \Phi_s \Rightarrow f(u) = 1$, or $f(u, v) = 1$, or $f(v) = 1$ (where the *or* is inclusive).

We say that a faulty router configuration *g explains* the local fault detection state Φ_s of source s, if $s : u - v \in \Phi_s \Rightarrow g(u) = 1$, or $g(v) = 1$.

We say that a faulty element configuration f *explains* the extended fault detection state Φ_s^e of source s, if

1. $s : u - v \in \Phi_s \Rightarrow f(u) = 1$, or $f(u, v) = 1$, or $f(v) = 1$.

2. $\sigma : u - v \in \Phi_S \Rightarrow f(\sigma) = 1$, or $f(u) = 1$, or $f(u, v) = 1$, or $f(v) = 1$.

We say that a faulty router configuration *g explains* the extended fault detection state Φ_s^e of source s, if

1. $s : u - v \in \Phi_s \Rightarrow g(u) = 1$, or $g(v) = 1$.

2. $\sigma : u - v \in \Phi_S \Rightarrow g(\sigma) = 1$, or $g(u) = 1$, or $g(v) = 1$.

4.2 A Pessimistic Estimate of the Number of Faulty Elements

From a pessimistic point of view, a source s can consider all elements that appear in its local fault detection state Φ_s as faulty. This faulty configuration explains the local fault detection state and its degree is

$$\beta_f = \left| \left(\bigcup_{s:u-v \in \Phi_s} s : u - v \right) \setminus s \right|$$

Assuming that all faulty routers and links appear in the local fault detection state of s, then β_f is also an upper bound on the number of faulty elements. Without making this assumption it is not possible to obtain an upper bound on the number of faulty elements since faulty routers and links may have not yet exhibited any misbehavior.

If s has obtained the detection states of other sources, then a pessimistic estimate of the number of faulty elements can be obtained by considering all elements that appear in the extended fault detection state Φ_s^e as faulty. This faulty configuration explains the extended fault detection state and its degree is

$$\beta_{f'} = \left| \left(\bigcup_{s:u-v \in \Phi_s} s : u - v \right) \bigcup \left(\bigcup_{\sigma:u-v \in \Phi_S} \sigma : u - v \right) \setminus s \right|$$

Note that $\beta_f \leq \beta_{f'}$ with the equality holding when Φ_S is empty. The disadvantage of estimating the number of faulty elements with $\beta_{f'}$, rather than β_f, is that even a small number of faulty sources can announce a large number of faulty nexuses and, therefore, the estimate will be unrealistic.

4.3 A Lower Bound from Local Fault Detection State

From an optimistic point of view, a source s can estimate the number of faulty elements by computing a faulty configuration that explains the local (or extended fault detection state) and whose degree is minimal. We will first describe how to obtain this estimate from the local fault detection state; the procedure for obtaining an estimate from the extended fault detection state can be found in the next section. We are going to show that the lower bound on the number of faulty elements, given local fault detection state, can be obtained by a solution of the minimum vertex cover problem [6] in a reduced form of graph G. In this section we will state the theorems; the proofs can be found in the Appendix.

We are given a graph representation $G(V, E)$ of the network, a source/vertex s in the graph, and its local fault detection state Φ_s.

Theorem 1. *It holds that*

$$\beta_{f^*} = \min_{\{f | f \, explains \, \Phi_s\}} \beta_f = \min_{\{g | g \, explains \, \Phi_s\}} \beta_g = \beta_{g^*}$$

This theorem states that the minimum degree of a faulty router configuration is equal to the minimum degree of a faulty element configuration and, thus, the latter can be computed by computing the former.

Definition 1 (Vertex Cover). *A* vertex cover *for G is a subset V^c of V such that, for each edge $(u, v) \in E$, at least one of u and v belongs to V^c.*

The following theorem specifies the reduced form of graph G whose minimum vertex cover is a faulty router configuration.

Theorem 2. *Consider the graph G_{Φ_s} that consists of all the edges, and their incident vertices, that belong to nexuses in Φ_s.*

1. If a faulty router configuration g explains Φ_s, then $\{u : g(u) = 1\}$ is a vertex cover of G_{Φ_s}.

2. *If $V^c_{\Phi_s}$ is a vertex cover of G_{Φ_s}, then the faulty configuration g in which $g(u) = 1, u \in V^c_{\Phi_s}$ and $g(u) = 0, u \notin V^c_{\Phi_s}$ explains Φ_s.*

Corollary 1. *The minimum vertex cover of G_{Φ_s} is a lower bound on the number of faulty elements in the network.*

The minimum vertex cover problem is an NP-hard problem that admits, however, efficient approximate solutions. In particular, it can be approximated within $2 - \frac{\log\log|V|}{2\log|V|}$ [6].

4.4 A Lower Bound from Extended Fault Detection State

If extended fault detection state is available at the source, then it is not hard to show similar to the above development that a faulty element configuration with minimal degree can be obtained by a solution of the *minimum set cover problem* [1], which is described below.

Definition 2 (Set Cover). *Given a collection C of subsets of a finite set U, a* set cover *of U is a subset $C' \subset C$ such that every element in U belongs to at least one member of C'.*

The minimum set cover problem is approximable within $min(\log|U|, f)$ [6, 26], where f is the maximum number of occurrences of some element of U in the subsets belonging to the collection C.

Given the extended fault detection state Φ^e_s of s, one could reduce it to a *minimum set cover problem*. Consider the set U which is constructed as follows: If $s : u - v \in \Phi_s$, then $\{u, v\} \in U$. Also if $\sigma : u - v \in \Phi_S$, then $\{\sigma, u, v\} \in U$. We then construct C in the following manner. For each router $r \in V$, there is a subset C_r that contains all nexuses containing r and also those nexuses that were identified by r as defective. Formally, $C_r = \{n_r | r \in n_r, n_r \in U\}$.

Corollary 2. *The size of the minimum set cover of U using the elements of C is equal to the degree of a minimum faulty configuration that explains the extended fault detection state.*

Note that the number of occurrences of an element $n_r \in U$ in the collection C is at most three. Therefore, the minimum set cover problem could be solved with a polynomial-time approximation algorithm that achieves a factor of three approximation. Also,

Corollary 3. *The degree of a minimum faulty configuration is approximable within a factor of three.*

5 Byzantine Detection Background II

In this section we provide additional background information on Byzantine detection that is necessary in the development that follows.

5.1 Elements of a Byzantine Detection Protocol

We now outline the mechanisms that are required to enable Byzantine detection. A protocol with Byzantine detection properties should be capable of identifying a misbehaving nexus (comprising of a pair of adjacent routers) when packets are tampered with or dropped. This basic requirement necessitates the following secondary requirements for protocols that guarantee Byzantine detection.

R1: The destination node needs to acknowledge the successful reception of packets. Drops could be identified by routers that are upstream from the drop location using timeouts. When a packet drop is detected, a fault announcement (FA) needs to be transmitted back to the source in order to pinpoint the failure location.

R2: Locations where packets are modified need to be identified precisely. If a packet is modified, the router that is immediately downstream needs to recognize the modification and drop the packet; the router that is upstream to the point of modification could then suffer a timeout and transmit an FA. Note that routers that are downstream from a Byzantine node cannot expect to reliably communicate FAs to the source due to the presence of the intervening Byzantine node.

R3: Consider a packet whose integrity verifies at some non-faulty node. If the packet remains unmodified, then it should pass the integrity checks at all other non-faulty downstream routers. In other words, a Byzantine node should not be able to modify the packet (or its authentication tags) in a manner such that it passes verification only at some subset of the non-faulty routers.

R4: ACKs and FA must be impractical to forge. ACK forgeries must be prevented so that malicious routers cannot deceive the source that the destination is receiving dropped packets. FA forgeries must be prevented so that malicious routers cannot deceive the source that faults are occurring in non-faulty nexuses.

R5: ACKs and FAs must be verified by every node that is upstream from the node generating it. Otherwise, ACK and FA forgeries will be detected only at the source, thereby preventing the detection of Byzantine nodes.

R6: If an ACK or FA verifies at one non-faulty router in the path, then it must verify at all other upstream non-faulty routers in the path. Otherwise, the adversary would gain the opportunity to discredit non-faulty nexuses by causing the FA to be deemed valid at a non-faulty router and invalid at its upstream router.

The above set of requirements imply the need for a mechanism that utilizes acknowledgments, timeouts, and fault announcements. In addition, the data packets, ACKs, and FAs need to have authentication tags that are carefully designed in order to satisfy the above-mentioned requirements. The Byzantine detection protocol of Avramopoulos et al. [3] proposes authentication structures that satisfies all of the requirements.

5.2 The Byzantine Detection Protocol of Avramopoulos et al.

In [2, 3], the source s shares secret keys with all the routers $n_1, \ldots, n_i, \ldots, n_m$ in the path that communication will be carried out. We denote $K_s^{n_i}$ the secret key shared between the source s and router n_i. The mechanism also requires a one-way hash function $h(\cdot)$. Given y and $h(\cdot)$ it is impossible to derive any x such that $h(x) = y$.

The source authentication tag (SAT) consists of message authentication codes (MACs), one for each downstream router. The MACs in the tag are computed sequentially from destination to first downstream router so that the MAC for a given router depends on the MACs of all routers that are downstream to it. This structure enables the mechanism to satisfy requirements R2 and R3.

For ACKs and FAs, the mechanism of [3] takes the following approach. Suppose that the sequence number of a packet is k and its source route is $< s, n_1, \ldots, n_i, \ldots, n_m >$, where s is the source and n_m is the destination. The source constructs m hash chains each of length three. The first element $r_i^0(k)$ of the hash chain for node i, $i = 1, \ldots, m$, is constructed by concatenating the sequence number k and the key $K_s^{n_i}$. The second and third elements $r_i^1(k)$ and $r_i^2(k)$ are constructed by applying a one-way hash function $h(\cdot)$ to the previous element. We will call k the *authenticator seed* or *seed*, $r_i^1(k)$ the *authenticator*, and $r_i^2(k)$ the *authenticator anchor* or *anchor*.

Subsequently the source announces with the packet the anchors, i.e., elements $r_i^2(k)$, which are protected by the SAT of the packet. Each recipient n_i is able to construct $r_i^0(k)$ by concatenating the seed with the secret key shared with the destination and then $r_i^1(k)$ by applying $h(\cdot)$. The latter element is used as the authenticator for the FA, if n_i is an intermediate router, or by n_m for the ACK.

6 Providing Arbitrary Downstream Feedback to the Source

We now consider the following problem: How can a source obtain an arbitrary piece of information from each of the routers along a path in a secure and efficient manner with Byzantine detection properties? Such a mechanism could be gainfully applied in many settings. It can be used, for example, to provide information of the delay that data packets experience in the routers of a path so as to mitigate the effects of maliciously introduced delay by adversarial routers [3]. In this paper, we are primarily interested in such a mechanism in order to develop an efficient data-forwarding protocol with Byzantine-detection properties (as discussed in Section 7).

We begin by discussing some alternatives for the design of such a mechanism. One possibility is to have the source first send along the path a query protected by the Byzantine detection protocol of Section 5.2 so that locations where the query is dropped could be identified. However, the hash elements used as authenticators for control messages (such as ACKs or FAs) are unsuitable for protecting the integrity of the query responses from the routers.

To address this limitation, one could consider the following approach. On receipt of the query, the destination appends to the ACK authenticator its *unauthenticated* response. On receipt of an ACK authenticator each router upstream to the destination first stores the responses of the routers downstream to it, then appends its own

unauthenticated response, and forwards the ACK. On receipt of the responses the source creates and forwards a new packet, called the *assertion packet*, that reflects the responses which are now protected by the SAT. On its receipt, downstream routers compare the responses that are reflected in the packet with the stored responses. If they agree, then the assertion packet is forwarded further. If they don't agree, then the corresponding router detects that either its immediate upstream nexus or the source are faulty and responds to this detection by dropping all subsequent packets that may arrive from the source in this path, thereby enabling the detection of the Byzantine behavior. Finally, the ACK from the destination could serve as the confirmation that the assertion packet reached the destination.

This protocol is however vulnerable to an attack according to which two malicious routers that are present in the path detour the packet and the ACK and append forged responses for the routers that have been intentionally bypassed. For example, in the path $< s, \ldots, x_1, u, v, x_2, \ldots, t >$, if x_1 and x_2 are malicious they can bypass routers u and v. This vulnerability could be addressed by using the following approach, albeit with increased computational overheads by using a cumulative ACK that comprises of hash value authenticators from each intermediate router.

After the receipt of the unauthenticated responses, the source constructs m hash chains each of length three, and then sends the third element of each hash chain along with the assertion packet. When an intermediate router receives the assertion packet, it checks whether the attached information is accurate. If the router receives the wrong information, it simply drops the packet, else it forwards the packet and waits for a partial cumulative response from the next router. If it gets such a response, it will append the authenticator, which is the middle element of its hash chain value, to the cumulative response. If there is a timeout, it starts with an empty cumulative response and appends its authenticator to it. The source can then identify till what point verification took place successfully.

Note however that the scheme described above requires a cumulative ACK, all of whose elements have to be verified by the upstream routers in order to satisfy requirement R5. This imposes a high overhead on intermediate routers, which we eliminate with the following novel design for packet authentication tags.

6.1 A Suitable Source Authentication Tag

In response to the aforementioned threat we create a source authentication tag that satisfies the following additional requirement for the handling of assertion packets.

R7: If a packet is verifies at a non-faulty router, every non-faulty router upstream to it has received the packet.

The SAT that we propose in order to satisfy the desired requirements is computed with a combination of MACs and hashes. Let $MAC(K, p)$ denote a MAC computed on message p with key K, let $h(p)$ denote a one-way hash computed on message p, and let $x|y$ denote the concatenation of strings x and y. Assume that communication is carried out in the path $s, n_1, \ldots, n_i, \ldots, n_m$, where s is the source and n_m is the destination.

The source initially computes one MAC for each downstream router: $MAC[n_i] \equiv MAC(K_s^{n_i}, p), i = 1, \ldots, m$, where p is the packet minus the SAT. Subsequently the source hashes the computed MACs and creates the destination's part of the SAT:

$$T[n_m] = h(MAC[n_1]| \cdots |MAC[n_m])$$

The intermediate routers' parts of the SAT are computed next:

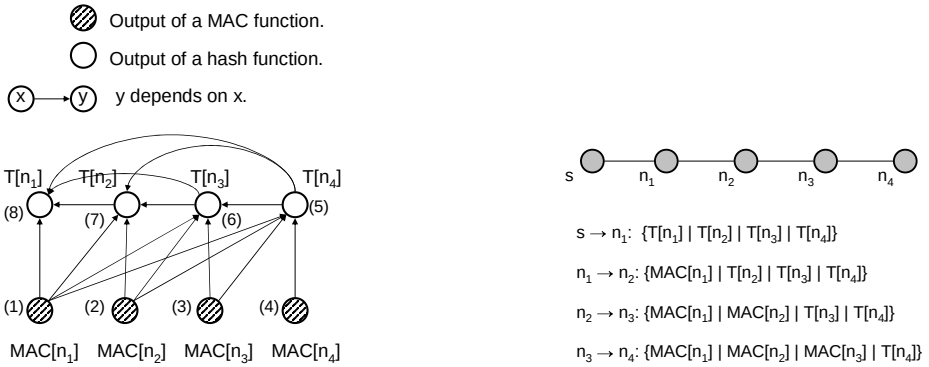
$$\begin{aligned} & \cdots \\ T[n_i] &= h(MAC[n_1]| \cdots |MAC[n_i]|T[n_{i+1}]| \cdots |T[n_m]) \\ & \cdots \\ T[n_1] &= h(MAC[n_1]|T[n_2]| \cdots |T[n_m]) \end{aligned}$$

A graphic representation of the computation of the SAT for the case that $m = 4$ is shown in Figure 1(a).

As the packet is propagated in the path, each router $n_i, i = 1, \ldots, m-1$ after verifying the authenticity of $T[n_i]$ replaces $T[n_i]$ with $MAC[n_i]$. This allows the SAT to be verifiable at downstream routers. The modifications in the SAT as the packet is propagated in a path where $m = 4$ are shown in Figure 1(b).

Requirement R3 is satisfied by the requirement from each $T[n_i], i = 2, \ldots, m$ to depend on all $T[n_j], j < i$. R7 is satisfied by the fact that $T[n_i], i = 2, \ldots, m$ is not verifiable unless $MAC[n_j], j < i$ has been disclosed.

The modification of the authentication tag of [3] in order to create this structure was inspired by the use of Merkle's hash tree in the protocol by Hu et al. in [12].



(a) **(b)**

Figure 1: The left of the figure contains a graphic representation of the computation of the SAT of Section 6.1 and the right of figure contains instances of the SAT as a packet is propagated in a path, for the case that $m = 4$.

7 The Nicephorus Byzantine Detection Protocol

In this section, we present the Nicephorus Byzantine detection protocol that achieves a significant performance improvement over previously proposed protocols without loss of detection accuracy.

In the Avramopoulos et al. protocol the source computes a hash chain of length three for each data packet and for each downstream router and announces the third element of each hash chain, i.e., the anchor, along with the source authentication tag. Each downstream router must, furthermore, recompute the second element of the chain (that will serve as the authenticator for the ACK or FA).

In contrast, a source in the Nicephorus protocol neither computes the aforementioned hash chains nor announces anchors for each data packet, which saves communication and computation overhead. Instead, downstream routers authenticate ACKs and FAs with elements of hash chains (of arbitrary length) that they have pre-computed. The Nicephorus Byzantine detection protocol therefore operates in two phases: an initialization phase for gathering anchors for hash chains and a data forwarding phase.

7.1 The Initialization Phase

For each path through which packets are going to be forwarded there is an initialization phase whose purpose is to build up state that will be subsequently used in a more efficient data packet forwarding phase. In the initialization phase, each downstream router is required to determine a hash chain whose elements are going to be used as authenticators for ACKs or FAs in the data packet forwarding phase and whose anchor must be securely announced to all of its upstream routers. As a first step, the source uses the mechanism of Section 6 to securely obtain the hash chain anchors from downstream routers. In particular, the source propagates through the path a query for hash chain anchors. All packets in the initialization phase, including this query packet, are protected as in Section 5.2 so that the locations of drops (or modifications) of these packets are detected.

As in Section 6, downstream routers append to the authenticator of the destination ACK for the query packet their unauthenticated responses and store the responses from downstream routers. On reception of the ACK, the source creates the protected (using the SAT of Section 6.1) assertion packet that reflects the unauthenticated responses. On approval of these responses, downstream routers further forward the assertion packet. However, if a router detects a discrepancy between the stored query responses and the query responses that the assertion packet carries, it will drop all subsequent packets that arrive in this path. This will cause the source to impose a large penalty on the corresponding faulty nexus upstream to this router, which will eventually be avoided.

On reception of an ACK to the assertion packet, the source is ensured of the authenticity of the received hash chain anchors. As a second step, the source announces the received anchors in a packet that is protected by the SAT of Section 6.1 and, on reception of an ACK for this packet, initiates the data packet forwarding phase.[1]

7.2 The Data Packet Forwarding Phase

In this section, we are going to use the elements of the hash chains that were precomputed in the initialization phase as the authenticators for ACKs and FAs. The efficient use of these elements requires addressing the

[1]Note that the name "data packet forwarding phase" is conventional. Data can be forwarded in the initialization phase as well.

following challenge: In order to gain the computational advantage of the hash chain, the authenticity of each hash element must be verified by computing exactly one hash. However, authenticators might be lost because of an innocuous or malicious drop and, therefore, subsequently released authenticators may require multiple hashes to verify. Furthermore, authenticators can be subject to replay. Naive handling of repeated authenticators might give malicious routers the advantage to discredit non-faulty elements by persistently causing the source to re-associate old innocuous faults with non-faulty elements using authenticator replay attacks.

The idea to address the aforementioned challenge is for downstream routers to *retransmit authenticators for ACKs and FAs until the source has acknowledged their receipt*. Such source acknowledgedments (SAs) are protected by the SAT of data packets (where the SAT is according to Section 5.2). After the reception of a source acknowledgement for an authenticator, the corresponding router will discard the old authenticator and release a new, yet undisclosed, authenticator in the next timeout expiration. Notice, however, that data packets that carry SAs might themselves be dropped before they reach the corresponding intended recipient of the SA. SAs are, thus, retransmitted until the source is ensured of their reception by the corresponding downstream routers.

The exact algorithm that Nicephorus uses is as follows: Each data packet contains a field, which we call the SA field, that consists of one bit for each downstream router. In the first packet after the initialization phase each bit is arbitrarily set. On receipt of an FA or an ACK, the source switches in the SA field the bit of the corresponding router from zero to one or from one to zero depending on its previous value. On receipt of a new SA, downstream routers take action depending on whether there is a transition in their corresponding bit of the SA:

- If there is a transition, the corresponding router discards the current authenticator and in the next timeout releases a new authenticator. Similarly, its upstream routers, expect a new authenticator for subsequent packets and will discard further use of the current authenticator as a replay.
- If there is no transition, the corresponding router retransmits the old authenticator after the next and subsequent timeouts (until instructed by the source to discard it). Similarly, its upstream routers, do not expect a new authenticator for the next control message and will accept the current authenticator as valid.

Note that in Nicephorus ACKs and FAs are no longer associated with any specific packet and they do not carry any sequence numbers; the expected authenticator for an ACK or FA is unambiguous at both the source and intermediate routers. Packet ordering and retransmission is the responsibility of Layer 4. Handling multiple outstanding packets is straightforward by, for example, partitioning the sequence number space to parts equal in number to the maximum number of packets that can be outstanding in the path and devoting one hash chain at each downstream router to each part.

As a final note, special care must be taken at the Nicephorus protocol when calculating the timeout values because of an attack that is made possible due to absence of sequence numbers in the ACKs and FAs. This attack as well as the corresponding countermeasure is the subject of Appendix B.

Security

After an authenticator is disclosed, it could be replayed by any malicious router to the router's upstream. However, by replaying such authenticator, a malicious router can only signal to the source the *actual* event of a drop at the downstream nexus of the intermediate router that the replayed authenticator belongs to or the *actual* event of a reception of a packet by the destination. Authenticators that have already reached the source should not be subject to replay, and this provision is made by the SA field in the SAT, which signals to downstream routers to discard old authenticators and release new ones. should a timeout fire.

As we previously mentioned, the SAT used for data packets allows *detours* of data packets by malicious routers. Therefore, SAs (that are carried in data packets) are subject to detour as well. Consider the following path:
$< s, \ldots, x_1, u, v, x_2, \ldots, t >$, where routers x_1 and x_2 are malicious, and routers u and v are not faulty. Any packet transmitted by s and source-routed in the aforementioned path can be detoured from x_1 to x_2 and not forwarded from x_1 to u as it is required. Now suppose that s acknowledges receipt of an authenticator by u but x_1 detours the packet. Since the corresponding bit of the SA field is retransmitted by s until a new authenticator is received by s, router x_1 cannot forward a new packet to u without also forwarding the SA.

Computation Overhead	Nicephorus 1	Nicephorus 2	Avramopoulos et al.	Awerbuch et al.
Source Authentication Tag	$2mM + 2mH$	$2mM$	$2mM$	$2mM + 2mE$
Fault Feedback Mechanism	$3mH$	mH	$3mH$	$2mM + 2(m-1)E$
Communication Overhead	Nicephorus 1	Nicephorus 2	Avramopoulos et al.	Awerbuch et al.
Source Authentication Tag	m^2S	$m^2S + c$	m^2S	m^2S
Fault Feedback Mechanism	$m(m+1)S$	mS	$m(m+1)S$	$\frac{1}{2}m(m+1)S$

Table 1: Comparison of the computation and communication overheads of Byzantine detection protocols.

Therefore, detouring packets in order to deprive routers from source acknowledgements does not pose a threat to the protocol.

7.3 Comparison with other Byzantine Detection Protocols

Table 1 compares the computation and communication overheads of Byzantine detection protocols. Nicephorus 1 and 2 are respectively the protocols in the initialization and data forwarding phase of Nicephorus, Avramopoulos et al. is the Byzantine detection protocol of [2, 3] and Awerbuch et al. is the Byzantine detection protocol of [4]. The "fault feedback mechanism" corresponds to the costs associated with ACK and FA generation, verification, and transmission. We denote by m the number of hops in a path, by M a MAC computation, by E an encryption or decryption, by H a hash computation, by S the size of the output of a MAC or hash computation, and by c the size of the source acknowledgement of the Nicephorus protocol. The computation and communication overheads are the total corresponding overheads in the path. We count the computation of the hash chains that are used for the authenticators for ACKs and FAs in the data forwarding phase in Nicephorus 2. The table shows a clear performance advantage of the data forwarding phase of Nicephorus (Nicephorus 2) over the other proposals.

8 Related Work

In a significant contribution, Byzantine detection was first proposed by Herzberg and Kutten [9] using an abstract model. In this model, it is prescribed that upon reception of a packet by an intermediate router or the destination, all upstream routers must have received the packet, and the paper notes that this property can be satisfied with cryptographic techniques. Nicephorus proposes one such technique that is utilized in an initialization phase. However, Nicephorus does not require this property to hold for normal packets, which results in more efficient cryptographic protection. Herzberg and Kutten proposed a *communication optimal* and a *time optimal* protocol as well as protocols that trade off communication and time optimality. The Nicephorus protocol makes use of both abstract protocols and employs a novel mechanism that switches between the communication optimal and the time optimal protocols based on an estimate of the degree of penetration of the adversary in the network.

In [2, 3], we proposed Byzantine detection protocols that are based on efficient symmetric cryptographic primitives and addressed issues such as replay and denial-of-service protection. Nicephorus is based on these Byzantine detection protocols, but improves their performance and, furthermore, contributes a novel mechanism to balance the per-packet performance with the recovery time capability of Byzantine detection protocols.

In [4], Awerbuch et al. propose Byzantine detection protocols that rely on MACs and encryption rather than the MACs and hashes that are utilized in Nicephorus. Awerbuch et al. also propose a cryptographic technique in the source authentication tag of data packets that requires these packets to take an intended route. As shown in Table 1, the initialization phase of Nicephorus is of comparable overhead to the protocol of Awerbuch et al., however, the data forwarding phase of Nicephorus incurs less overhead since it utilizes a more efficient source authentication tag and a more efficient technique to receive feedback on faults.

In [19], Padmanabhan and Simon propose a *secure traceroute* tool for the purpose of detecting locations of malicious failures in the forwarding of data packets. Secure traceroute verifies the correct behavior of a route incrementally. At each step the behavior of a new router is verified. Secure traceroute bears some similarity with Byzantine detection; the objective is to detect faulty links using feedback from downstream routers and authentication. There are some important differences however. First, secure traceroute packets must look indistinguishable from normal traffic for routers whose behavior has not yet been verified as non-faulty. Byzantine detection protocols, on the other hand, do not require concealment of packets. Second, secure traceroute is designed to work with hop-by-hop routing whereas Byzantine detection proposals involve source routing. Third,

secure traceroute is intended for environments with a relatively limited number of adversarial routers, whereas Byzantine detection can potentially provide recovery in environments where the adversary has achieved significant penetration.

Bradley et al. [7] propose a protocol for detecting and avoiding routers that are dropping or mis-routing packets. The basic idea of this protocol is for routers to test for "the conservation of flow" principle by measuring the volume of flows that enter and leave their incident links and comparing the measured flows in a global coordination phase. A limitation of this approach is that protection against packet modifications is not addressed.

In [16,17], Mizrak et al. propose an approach for the protection of routing that is based on a traffic validation component that monitors traffic characteristics and looks for anomalous behavior, a distributed detection component that coordinates the traffic monitors and detects faulty routers or groups that contain faulty routers, and a response component that takes countermeasures against detected faulty routers (or groups). Two important characteristics of the proposed detection components [16] that are not found in Byzantine detection protocols are time synchronization that facilitates the comparison of collected traffic, and Byzantine consensus [14] which ensures uniform decisions on detected faulty network paths.

One approach taken in the literature for data packet forwarding protection against malicious routers is *multipath routing*, as in the work by Perlman [22] and Papadimitratos and Haas [21]. Multipath protection can provide detection at the granularity of a *path* and incurs less overheads than Byzantine detection, but it does not provide sufficient information by itself to enable fast recovery procedures as the paths between source and destination can be exponentially many. We would like to note here that Perlman also proposed in [22] an approach to fault diagnosis that can be seen as a precursor to Byzantine detection protocols.

Listen and Whisper is proposed in [25] as a BGP (Border Gateway Protocol) protection mechanism that combines use of cryptographic techniques that are incorporated in the BGP protocol (Whisper) and data-packet-flow monitors that verify whether the routes obtained by Whisper are operational (Listen). Listen and Whisper introduces the interesting concept of detection and containment of faulty ASs that attack the routing protocol. Such attacks are not within the scope of this paper. Listen signals the occurrence of problems in the data plane but does not pinpoint these problems (in contrast with the goal of Byzantine detection).

Secure topology and route discovery protocols such as [8, 10–13, 18, 20, 24] and others are complementary to data packet forwarding protection mechanisms in order to improve the resilience of the routing function.

9 Conclusion

Routing attacks can create severe service disruptions. These attacks can be targeted at both the routing protocol and the data forwarding mechanism. Protecting data forwarding is of equal importance to the protection of the routing protocol. A secured routing protocol does not, for example, prevent the adversary from announcing fictitious links between distant routers that it controls and subsequently dropping the increased volume of traffic that it receives. Byzantine detection is a promising tool to counter such an adversary by identifying the adversarial locations at the granularity of a link (or nexus) so that these locations can be subsequently bypassed.

Byzantine detection protocols allow trade-offs between the recovery capability that they offer and the overhead that they incur. Based on this observation, we proposed a mechanism that can dynamically balance these parameters. This mechanism estimates the threat that the adversary poses to the packet delivery service by computing the minimum number of adversarial elements that explains the fault detection state and activates protocols with different properties regarding recovery capability and overhead based on this estimate. We are investigating algorithms to compute faulty configurations that minimize cost functions more general than the cardinality of the set of adversarial elements, which we presented in this paper.

We also made two primary contributions to Byzantine detection protocols. The first contribution is a mechanism that can be used to query the routers of a path for router-specific information. This mechanism has Byzantine detection capabilities so that the locations where the query or its responses might be modified or dropped can be detected. Our second contribution is an efficient data forwarding protocol with Byzantine detection capability that makes use of the query mechanism in order to bootstrap an efficient data forwarding phase.

References

[1] G. Ausiello, A. D'Atri, and M. Protasi. Structure preserving reductions among convex optimization problems. *J. Comput. System Sci.*, 21:136–153, 1980.

[2] I. Avramopoulos, H. Kobayashi, R. Wang, and A. Krishnamurthy. *Amendment to Highly Secure and Efficient Routing*, Feb. 2004. Addendum to [3].

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

[4] 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.

[5] W. Cheswick, S. Bellovin, and A. Rubin. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison Wesley, second edition, 2003.

[6] P. Crescenzi and V. Kann. *A Compendium of NP Optimization Problems*. http://www.nada.kth.se/˜ viggo/problemlist/compendium.html.

[7] K. Bradley et al. Detecting disruptive routers: A distributed network monitoring approach. *IEEE Network Magazine*, Sept./Oct. 1998.

[8] G. Goodell, W. Aiello, T. Griffin, J. Ioannidis, P. McDaniel, and A. Rubin. Working around bgp: An incremental approach to improving security and accuracy of interdomain routing. In *Proc. Network and Distributed System Security Symposium, NDSS '03*, San Diego, CA, Feb. 2003.

[9] A. Herzberg and S. Kutten. Early detection of message forwarding faults. *SIAM J. Comput.*, 30(4):1169–1196, 2000.

[10] Y.-C. Hu and A. Perrig. Spv: A secure path vector routing scheme for securing bgp. In *Proc. ACM SIGCOMM 2004*, Portland, Oregon, Sep. 2004.

[11] 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.

[12] 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.

[13] Stephen Kent, Charles Lynn, and Karen Seo. Secure border gateway protocol (secure-bgp). *IEEE Journal on Selected Areas in Communications*, 18(4):582–592, Apr. 2000.

[14] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[15] B. Leighton. Personal communication, Dec. 2003.

[16] A. Mizrak, K. Marzullo, and S. Savage. Detecting malicious routers. Technical Report CS2004-0789, University of California at San Diego, Department of Computer Science, May 2004.

[17] A. Mizrak, K. Marzullo, and S. Savage. Fault-tolerant forwarding in the face of malicious routers. In *Proc. 2nd Bertinoro Workshop on Future Directions in Distributed Computing*, Bertinoro, Italy, Jun. 2004.

[18] S. Murphy and M. Badger. Digital signature protection of the ospf routing protocol. In *Proc. Symposium on Network and Distributed System Security, NDSS '96*, San Diego, CA, 1996.

[19] V. Padmanabhan and D. Simon. Secure traceroute to detect faulty or malicious routing. In *Proc. ACM SIGCOMM First Workshop on Hot Topics in Networks (HotNets-I)*, Princeton, NJ, Oct. 2002.

[20] 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.

[21] 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.

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

[23] J. Rexford. Personal communication, Apr. 2004.

[24] 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.

[25] L. Subramanian, V. Roth, I. Stoica, S. Shenker, and R. Katz. Listen and whisper: Security mechanisms for bgp. In *Proc. First Symposium on Networked Systems Design and Implementation (NSDI'04)*, San Francisco, CA, Mar. 2004.

[26] V. V. Vazirani. *Approximation Algorithms*. Springer Verlag, 2001.

A Proofs of the Theorems in Section 4

Lemma 1. *Given any Φ_s, there is always an f that explains it.*

Proof. The faulty element configuration that considers all elements in the set $\left(\bigcup_{s:u-v \in \Phi_s} s : u - v \right) \setminus s$ as faulty explains Φ_s. □

Lemma 2. *Given any Φ_s, there is always a g that explains it.*

Proof. The faulty router configuration that considers all routers in the set $\left(\bigcup_{s:u-v \in \Phi_s} (s : u - v \setminus (u, v)) \right) \setminus s$ as faulty explains Φ_s. □

Theorem 1. *It holds that*

$$\beta_{f^*} = \min_{\{f | f explains \Phi_s\}} \beta_f = \min_{\{g | g explains \Phi_s\}} \beta_g = \beta_{g^*}$$

Proof. Since $\{g | g explains \Phi_s\} \subset \{f | f explains \Phi_s\}$ it holds that $\beta_{f^*} \leq \beta_{g^*}$. We will next show that $\beta_{g^*} \leq \beta_{f^*}$.

Consider a nexus $s : u - v \in \Phi_s$. We will show that if $f^*(u, v) = 1$, then we can construct a configuration $\hat{f}$ that explains Φ_s and in which $\hat{f}(u, v) = 0$ and $\beta_{f^*} = \beta_{\hat{f}}$. Assume initially that $\hat{f} = f^*$.

1. Assume that $f^*(u) = 1$, $f^*(u, v) = 1$, and $f^*(v) = 0$. If $\hat{f}(u) = 1$, $\hat{f}(u, v) = 0$, and $\hat{f}(v) = 1$, then $\hat{f}$ explains Φ_s and $\beta_{f^*} = \beta_{\hat{f}}$.
2. Assume that $f^*(u) = 0$, $f^*(u, v) = 1$, and $f^*(v) = 1$. Symmetrical to the above.
3. Assume that $f^*(u) = 0$, $f^*(u, v) = 1$, and $f^*(v) = 0$. If $\hat{f}(u) = 1$, $\hat{f}(u, v) = 0$, and $\hat{f}(v) = 0$, then $\hat{f}$ explains Φ_s and $\beta_{f^*} = \beta_{\hat{f}}$.

By repetitively applying these steps we end up with a faulty router configuration $\hat{f}$. Therefore, $\beta_{g^*} \leq \beta_{\hat{f}} = \beta_{f^*}$, which proves the theorem.

□

Definition 1 (Vertex Cover). *A* vertex cover *for G is a subset V^c of V such that, for each edge $(u, v) \in E$, at least one of u and v belongs to V^c.*

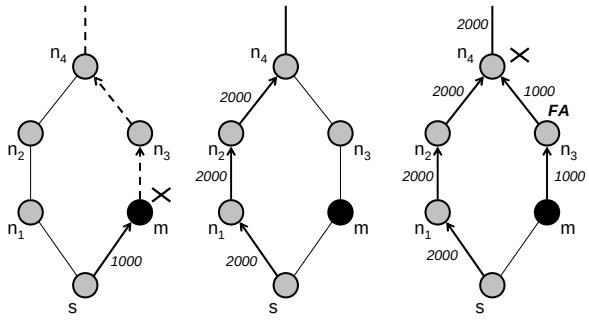


Figure 2: An example of a faulty intermediate router (m) causing a non-faulty router (n_3) to reveal its FA authenticator about non-faulty nexus ($n_3 - n_4$) assuming that the source calculates timeout values per reference [3]. Router m stores and drops the packet with sequence number 1000 and reinserts it in the network, when the source inserts the packet with sequence number 2000. Router n_4 will drop the packet with sequence number 1000 as a replay and the timeout at n_3 will fire. Thus, n_3 will disclose its FA authenticator without any innocuous fault having occurred at nexus $n_3 - n_4$.

Theorem 2. *Consider the graph G_{Φ_s} that consists of all the edges, and their incident vertices, that belong to nexuses in Φ_s.*

1. *If a faulty router configuration g explains Φ_s, then $\{u : g(u) = 1\}$ is a vertex cover of G_{Φ_s}.*
2. *If $V^c_{\Phi_s}$ is a vertex cover of G_{Φ_s}, then the faulty configuration g in which $g(u) = 1, u \in V^c_{\Phi_s}$ and $g(u) = 0, u \notin V^c_{\Phi_s}$ explains Φ_s.*

Proof. Regarding (1): If g explains Φ_s, then $s : u - v \in \Phi_s$ implies that $g(u) = 1$ or $g(v) = 1$. Since this holds for all $s : u - v \in \Phi_s$, then for every $(u, v) \in E_{\Phi_s}$ (the set of edges of G_{Φ_s}) $u \in V_{\Phi_s}$ or $v \in V_{\Phi_s}$.

Regarding (2): If $V^c_{\Phi_s}$ is a vertex cover, then $s : u - v \in \Phi_s$ implies that $u \in V^c_{\Phi_s}$ or $v \in V^c_{\Phi_s}$ and, thus, $g(u) = 1$ or $g(v) = 1$ respectively. Since this holds for all $s : u - v \in \Phi_s$, g explains Φ_s. □

Corollary 1. *The minimum vertex cover of G_{Φ_s} is a lower bound on the number of faulty elements in the network.*

Proof. By Theorem 1 the degree of a minimum faulty element configuration that explains Φ_s is equal to the degree of a minimum faulty router configuration that explains Φ_s. By Theorem 2 a faulty router configuration explains Φ_s if and only if it is a vertex cover of G_{Φ_s}. Therefore, the minimum vertex cover of G_{Φ_s} is the minimum faulty router configuration that explains Φ_s and, thus, a minimum faulty element configuration that explains Φ_s. □

B Setting Timeout Values in the Nicephorus Byzantine Detection Protocol

On transmission of a packet, the corresponding router (either a source or an intermediate router in a path) must set a timeout to receive an ACK or an FA. The timeout value is set as the worst-case round-trip-time from the corresponding router to the destination. As [3] points out, it is only safe to make the round-trip-time calculation at the source. The reason is that information about overlap between simultaneously used paths, which affects the timeout value because of queuing delays, is not readily available at intermediate routers. In [3] the worst-case round-trip-time is calculated by taking into account all packets that have been transmitted by the corresponding source and whose timeout is still pending.

In Nicephorus, the setting of timeout values must take into consideration a potential attack that does not apply to [3] according to which a non-faulty router is forced to reveal its authenticator about a non-faulty nexus without a prior innocuous drop at that nexus, by an exploitation of the timeout calculation mechanism. An example of this attack in shown in Fig. 2. In the example, malicious router m forces router n_3 to reveal its authenticator without a prior innocuous drop at nexus $n_3 - n_4$ by first dropping and then replaying the packet with sequence number 1000 after the source has inserted the packet with a higher sequence number (2000 in the example).

This attack is successful in the example because router n_3 deems the packet with sequence number 1000 valid whereas router n_4 deems that packet invalid. In [3] the FA would carry the sequence number of the packet that it pertains to but in Nicephorus FAs do not carry sequence numbers. Therefore, if the source attempts to insert new packets to the path on the right, then m can replay the FA and inaccurately associate a fault with non-faulty nexus $n_3 - n_4$.

This attack can be prevented if, in the timeout calculations, the source considers pending all unacknowledged packets in the reserved parts of the sequence number space that may be considered valid at the routers of the corresponding paths. This information is readily available at the source based on the destination ACKs that it receives. In the example of Figure 2 the attack will not be successful if the source considers the packet with sequence number 1000 pending when calculating the timeout values for the packet with sequence number 2000.