More on Castor: the Scalable Secure Routing Protocol for Ad-hoc Networks

LSIR-REPORT-2009-002

Wojciech Galuba, Panos Papadimitratos, Marcin Poturalski, Karl Aberer
Ecole Polytechnique Fédérale de Lausanne (EPFL)
firstname.lastname@epfl.ch

Zoran Despotovic, Wolfgang Kellerer
DOCOMO Euro-Labs, Munich, Germany
lastname@docomolab-euro.ch

***Abstract*—Wireless ad hoc networks are inherently vulnerable, as any node can disrupt the communication of potentially any other node in the network. Many solutions to this problem have been proposed. In this paper, we take a fresh and comprehensive approach, simultaneously addressing three aspects: security, scalability and adaptability to changing network conditions. Our communication protocol, Castor, occupies a unique point in the design space: it does not use any control messages except simple packet acknowledgements, and each node makes routing decisions locally and independently of other nodes without exchanging routing state with them. This novel design makes Castor resilient to a wide range of attacks and allows it to scale to large network sizes and to remain efficient under high mobility. We compare Castor against four representative protocols from the literature. Our protocol achieves up to two times higher packet delivery rates, particularly in large and highly volatile networks, incurs no or only limited additional overhead and it is able to survive more severe attacks and recovers from them faster.**

I. Introduction

The peer-to-peer, distributed operation of ad hoc networks, as well as the nature of wireless communication pose significant security challenges. Without appropriate security mechanisms, the adversary can affect or even control the self-organizing operation of the network, and degrade or completely prevent communication. Thus, a fundamental security objective in such systems is thus to *secure communication*. Secure communication protocols should be able to maintain acceptable data delivery rates under all feasible attacks. *Secure route discovery* is essential for solving this problem. Without it, the data sent across the routes the adversary manipulated would never be received at their destinations. However, securing the route discovery is not sufficient:adversaries can always become a part of the routes by behaving correctly during the route discovery, and then strategically disrupt communication (e.g., drop or corrupt packets) once the routes are being used.

It is thus necessary, as some protocols surveyed in §II do, to utilize a *secure data transmission* protocol on top of secure route discovery. Secure data transmission protocols correlate data delivery failures with specific routes or network areas, possibly controlled by the adversary. Then, they reroute the traffic, to avoid the adversary and reestablish reliable communication.

This approach was shown to be effective, especially if sufficiently rich connectivity information is available. Simultaneous use of multiple paths, redundancy in transmissions, and end-to-end secure feedback allow for quick route convergence [1]. However, because of system constrains, there may not be enough bandwidth available for multi-path data transmission. When resources are scarcer, the generally applicable solution is a single-path secure communication protocol: sending data and feedback across a single path, and switching to another path once the current one is deemed unreliable.

Would such a solution remain efficient and effective in large and highly volatile networks, even in the presence of powerful adversaries? Consider networks that are open, mobile and can grow in size, with the subset of nodes supporting a given single-path flow constantly changing. Managing the variability, avoiding the faulty and adversarial nodes, while sustaining reliable communication is a challenge.

Our *Continuously Adapting Secure Topology-Oblivious Routing (Castor)* addresses exactly this problem. Each node keeps track of the reliability of its neighbors only; none of the local state is ever exchanged over the network; packet sizes do not carry routes, thus their length does not grow with the network size; each node operates fully autonomously, making routing decisions independently of other nodes and without knowing the network topology beyond its local one-hop connectivity. These features make Castor *scalable*. Moreover, in-network routing state allows Castor to *rapidly adapt* to a wide range of faults, malicious and benign, even under an *overwhelming adversarial presence*. Finally, the minimal exchange of information between the nodes implies there is little need for authenticating it or securing its transmission; this enables Castor to operate under the *simplest*, among those in the literature, *trust assumptions*.

Our extensive comparative evaluation shows that Castor outperforms four other protocols (SRP/SSP, Sprout, SEAD, and trivially the non-secure AODV), with significant advantages:

- Castor consistently achieves up to *40% higher packet delivery rate without any additional overhead*
- It recovers at least *twice as fast* as the other protocols
- It is the *only one* among the five evaluated here that achieves *full recovery* from the wormhole attack without the help of a secure neighbor discovery protocol.

Equally important, Castor maintains its advantage as the network *scales* and mobility increases: for example, in a 400 node network with 80 black-holes and continuous mobility, Castor achieves, with a mild overhead increase, a consistent 60% packet delivery rate, *double* that of other protocols.

In summary, our main **contributions** in this paper are:

- Castor, a scalable secure communication protocol that is highly resilient to a wide range of attacks and benign faults
- an extensive comparative performance evaluation with four other protocols, under various attacks, mobility and network size settings.

What sets Castor apart is its fundamentally novel approach, among secure communication protocols, and its versatility, in spite its simplicity. Castor is the first protocol to demonstrate robustness against such a large spectrum of attacks and for a wide range of network scales. Our comprehensive comparative evaluation of secure communication protocols is also the first of its kind.

In the rest of the paper, we first discuss related work (§II) and give an overview of Castor. Then, we define the system and adversary models (§IV) and present in detail the functionality of Castor (§V). We analyze its security (§VI) and evaluate the performance (§VII) before we discuss open questions (§VIII) and conclude.

This technical report is an extended version of [2]. The additional material consist in: the pseudo-code of Castor; an extended cryptographic scheme section, including an alternative, public key scheme; an extended performance evaluation section, including the grayhole attack and a summary; and a more detailed overview of interesting open questions in §VIII.

II. Related Work

Secure ad hoc networking protocols address two main issues: (a) secure route discovery, to prevent attacks on the disseminated routing information, and (b) secure data transmission, to ensure data delivery. Most proposals in the literature considered the first issue only or assumed the second one is addressed by upper layer protocols; few ones considered both issues.

Secure route discovery. SRP [3] is an on-demand protocol: it floods in a controlled manner a route request (RREQ), with intermediate nodes each appending its identifier. The destination returns route reply (RREP) packets strictly across the reverse of the path accumulated in the RREQs. End-nodes can authenticate each other and their RREQ and RREP packets; intermediate nodes do not need to authenticate traffic from end-nodes. Ariadne [4] follows the same principle, but it authenticates intermediate nodes at the end-nodes. This increases the trust management complexity and overhead, in return for stricter identification of the intermediate nodes at the source. endairA [5] takes essentially the same approach, utilizing only public key cryptography, and offers increased resilience to attacks.

SRP, Ariadne, and endairA provide the entire discovered route (connectivity information) to the source node, and the same is true for link state protocols such as SLSP [6]. In a different category, implicit route discovery protocols [7] provide each node with the next hop towards the destination: ARAN [8] discovers a single route, based on the first-returning RREP at the source; S-AODV [9] provides security for AODV [10], authenticating its RREQ, RREP and route error packets; and SEAD [11] protects distance-vector calculations from distance decrease, using symmetric key cryptography (while ARAN and S-AODV use digital signatures).

Secure data transmission. SSP [12] is a secure single-path protocol that relies on an end-to-end security association; it transmits packets across a route calculated over the connectivity the underlying route discovery provides (typically, protocols such as SRP). The destination validates received data and responds with acknowledgements; if not, the source detects a packet loss. The route rating is increased each time an acknowledgement is received, and it is reduced when a timeout occurs (no ACK); once the rating drops below a threshold, the route is discarded and the source switches to another one (invoking a new route discovery if needed). SSP is robust to any attack (e.g., wormholes, tunnels, other collusion attacks) that causes a packet to be dropped; if so, the route is discarded.

Sprout [13] is a protocol that source-routes data across a single path chosen among many alternative ones. Those paths are calculated over the topology view a secure link state discovery protocol offers, with nodes broadcasting link state updates across the network. In order to be resilient against colluding adversaries that advertise fictitious links, Sprout introduces mechanisms that prevent the pollution of the network link state view. Routes are generated and utilized probabilistically, acknowledgements are returned by the destination, and routes deemed operational are re-used while new alternative ones are explored. The link-state operation requires that any node can identify all other nodes at all times. SSP and Sprout are the two protocols closer to our Castor.

Other related schemes. ODSBR[14] discovers routes reactively, it updates link weights based on their behavior observed at the sources, and when communication reliability drops below a threshold, it augments data packets with probes to identify the wrong-doer. ODSBR requires that the source knows all nodes in the network. It can maintain reliability across a route above a threshold unless two or more colluding attackers are part of the route [13]. Beyond security protocols, reputation, and remuneration-based schemes have been considered [15]. All these schemes are complex and costly (e.g., requiring a full trust graph, long observation periods), or they can be effective only against rational adversaries, or they can be susceptible to attacks that incriminate correct nodes. Finally, a note on the so called ant-based routing protocols [16], [17], [18] which somewhat resemble Castor: they do not have an explicit route discovery phase and use the ”acks” to positively reinforce paths (by analogy to phermone traces). However, none of them considered security.

Comparison to Castor. In brief, Castor extends over secure route discovery, as it falls in the category of the comprehensive solutions that secure data transmission too; e.g., ODSBR, SRP plus SSP or SMT, and Sprout. Compared to those, it introduces significant differences. Concisely: (i) Routes need not be attached to packets, thus packets do not grow in length with the network size (and thus route length), (ii) there are no route discovery control packets, only data and acknowledgements, (iii) the communication reliability information is kept locally at each node in the network, not at the source, (iv) Castor does not seek to identify and exclude attackers, and (v) it relies on the simplest trust management assumptions (same as those of the SRP-SSP combination).

III. Protocol design overview

Castor operates as follows: when the source sends a packet, intermediate nodes forward it until it reaches its destination, which then responds with an acknowledgement that follows the reverse path back to the source. The basic design elements and ideas behind Castor are discussed next in this section.

Learning from failures. Nodes locally keep per-flow-and-next-hop reliability metrics (§V-D), which are updated constantly, based on the arriving acknowledgements indicating success and the acknowledgement timeouts indicating failure. These metrics are used to select the most reliable next hop for each incoming packet. If no reliable next hop exists, or if the recorded history is insufficient for the node to make a choice, the packet is locally broadcasted. Each node decides whether to unicast or broadcast independently (§V-C).

Reliability as the primary performance metric. Protocols that minimize the number of hops or the round-trip time, are susceptible to an attacker advertising shorter routes or setting up wormholes or tunnels to attract traffic and then drop passing data packets. To be robust against such attacks, Castor uses reliability as its primary metric. Since the routing is reliability-driven, Castor is able to detect and react to all causes of packet loss, independent of their nature, be it benign or adversarial.

Response time as the secondary performance metric. For performance reasons, Castor keeps routes short by giving preference to the first neighbor responding with an acknowledgment during the route discovery. However, this neighbor can be routed around if it turns out to be unreliable.

Routing and route discovery as a single process. There are only two message types in Castor: the payload-carrying packets and the acknowledgements. When a packet arrives, from the application layer of the source node or a previous hop, the best next hop is selected based on the kept performance history. When the history is insufficient or indicates likely failure then the protocol seamlessly switches to broadcasts, which serve the function of route discovery.

Emergent reliable routes. With every node estimating and acting on local reliability metrics, Castor is able to locally route to avoid unreliable neighbors. This results in fast global convergence to reliable routes and efficient continuous adaptation under mobility.

Local repair. In contrast to some other protocols, adversarial or benign failures do not always cause the costly network-wide floods to search for better routes. Most of the time, a Castor node has another reliable neighbor to switch to when one neighbor fails. It resorts to broadcasting only when facing severe failures. In most cases, a brief cascade of local broadcasts reaches a part of the network with reliable next hops and unicasting resumes.

Secure, isolated routing state. Nodes make routing decisions independently, based only on locally accumulated neighbor reliability metrics. In other words, nodes are oblivious to any network connectivity information beyond the local neighbors. No routing state is ever exchanged between nodes, which removes the problem of securing the information exchange. State locality and minimal control traffic are also key to Castor's scalability (§VII-D).

Routing state is stored on a per-flow, not per-destination, basis. A cryptographic scheme ensures that only the packets coming from the flow's source and the acknowledgments coming from the flow's destination can influence the routing state for that flow (§V-B). Despite relying on simple trust assumptions, these mechanisms provide a strong protection against routing state pollution by the adversary.

Immutable messages. All message fields in data and acknowledgement packets are immutable, i.e. they do not change as the message is forwarded across the network. This removes the problem of preventing field manipulation by on-route adversaries, e.g., ensuring that the hop count in route responses cannot be decremented.

Resource exhaustion countermeasures. To prevent resource exhaustion attacks and unnecessary flooding when destinations are unreachable, Castor uses a flood rate-limiting mechanism at each node (§V-E). Neighbors that cause too many packet floods without subsequent acknowledgements are throttled.

IV. Assumptions

A. System Model

We consider a wireless ad hoc network composed of static or mobile *nodes*: computing platforms with wireless transceivers that have a limited communication range. Nodes communicate directly over the wireless channel with their *neighbors*. Nodes assist the other nodes with communication across multiple links (hops). Each node has a unique identity, which can be cryptographically validated if needed. Nodes that conform to system protocols are *correct*, and those that deviate from them are *adversarial*.

Cryptography. Castor requires that for each pair of end nodes, a source s and a destination d, that wish to communicate securely across the network, either s and d share a pre-established symmetric key $K_{s,d}$ or s knows the public key K_d of d. Furthermore, we assume d is able to verify the integrity of the messages sent by s. We also assume that any two correct neighboring nodes can establish a shared secret symmetric key, to authenticate their communication. Neighbor-to-neighbor keys can be established and authenticated in a number of ways, depending on the system instantiation, e.g., through key transport or agreement. Authentication can be performed with the help of local channels, passwords, certificates etc. For example, a certified public key can serve as the verifiable unique identity of a node. Moreover, each correct node can authenticate messages it broadcasts at the data link layer, utilizing symmetric-key schemes such as [19].

Neighbor discovery. Neighbors are discovered by a simple mechanism, such as beaconing. We do *not* require a secure neighbor discovery protocol, which would prevent the adversary from convincing two non-neighbor nodes that they are neighbors [20].

B. Adversary Model

The adversary controls a number of adversarial nodes, which can be *internal* or *external*. The internal nodes are equipped with the same cryptographic material as the correct nodes. For example, a compromised but previously correct node can become an internal adversary. A single adversarial node can appear as multiple network nodes, utilizing multiple compromised identities and cryptographic keys.

An adversarial node can arbitrarily deviate from the protocol definition. In particular, it can drop, modify, and replay any message. The adversary is, however, computationally bounded and cannot break cryptographic primitives. If beneficial to the attacker, an adversarial node can correctly follow the protocol for any period of time. Adversarial nodes can also act in coordination and mount collusion attacks. Moreover, adversarial nodes can communicate across large distances using fast out-of-band communication links (typically used to mount e.g., *wormhole* and *tunnel* attacks) and jam communication.

The objective of the adversary we consider here is *denial of service*: to prevent communication or in other words to prevent messages from being delivered. In the rest of the paper, unless stated otherwise, we are concerned with the strongest variant of adversaries, internal and colluding. We do not seek to thwart any adversarial behavior that does not result in packet loss. In particular, we do not address the problem of preventing traffic interception, eavesdropping or analysis.

C. Metrics

We focus exclusively on flows between correct source-destination pairs. The primary performance metric is the *packet delivery rate* (PDR). More precisely, we are interested in the *network-layer PDR*. We want to capture the raw network performance in the presence of adversaries, without using any packet retransmission schemes, either at network or upper layers. Further, we are interested in the *bandwidth utilization* per delivered packet.

V. The protocol

A. Message specification

Castor uses two types of messages: PKTs and ACKs. The data packet, **PKT**, is a tuple $(s, d, H, b_k, f_k, e_k, M)$: s and d are the source/destination identifiers; H is the *flow identifier (id)*; b_k is the *PKT id*; f_k is the *flow authenticator*, used for verifying that the PKT belongs to flow H; e_k is an *encrypted ACK authenticator*. Finally, M is the payload, which typically includes an additional integrity protection mechanism.

The acknowledgment packet, **ACK**, has only one field a_k, an *ACK authenticator*, which is used for verifying that the corresponding PKT was delivered to the destination.

B. Cryptographic mechanisms

To ensure the correct flow state updates, the PKT and ACK fields need to satisfy two properties:

- First, an ACK a_k should only be received by an intermediate node if the destination has indeed received the corresponding PKT b_k. More precisely:
  (A1) given a_k and b_k, any intermediate node can verify that the ACK authenticator a_k corresponds to PKT b_k,
  (A2) given H, PKT b_k and e_k, as well as any number of other correct PKTs and ACKs from flow H (i.e., b_j, e_j, f_j, and a_j, st. $j \neq k$), only the destination can recover a_k.

  To satisfy this requirement, one can choose a_k to be a random nonce freshly generated by the source, set $b_k = h(a_k)$ (where h is a cryptographic hash function), and let e_k be an encryption of a_k either with the symmetric key K_{sd} shared between sa and d or the public key K_d of the destination. We assume the former for the remainder of this subsection.
- Second, no node except the source of flow H should be able to generate a PKT b_k that would be verified as belonging to H. We emphasize that we *do not* require an intermediate node to verify that all PKTs originate from some particular source – which is impossible as in our setup the source does not pre-share keys with intermediate nodes. Rather, an intermediate node verifies that all PKTs originate from the same, but arbitrary source. A precise statement of this property is:
  (B1) given H, b_k and f_k, any intermediate node can verify that PKT id b_k corresponds to flow H,
  (B2) given H, and any number of correct PKTs and ACKs from flow H (i.e., b_j, e_j, f_j and a_j for $j \neq k$), only the source that originated H can generate a new b_k, f_k pair that verifies as belonging to H.

There are many cryptographic schemes that can achieve the properties (A1), (A2), (B1) and (B2); we present here an efficient solution based on Merkle hash trees, as well as an alternative public-key scheme. The latter removes the minor inconvenience of flow restarting, as we explain below. The pseudocode of the Castor protocol given in Fig. 2, uses the former scheme.

1) Merkle tree scheme:

PKT generation. For each flow that the source s wants to send to a destination d, the source pre-generates: (i) a set of random nonces, $a_1, \dots, a_w$, the ACK authenticators, (ii) a corresponding set of PKT ids $b_k = h(a_k)$, where h is a cryptographic hash function and (iii) a Merkle hash tree with $h(b_1), \dots, h(b_w)$ as leaves. The root of this tree becomes the flow id H. The pre-generated values are then used when sending the k-th PKT $(s, d, H, b_k, f_k = [x_1, \dots, x_l], e_k = E_{K_{sd}}(a_k), M)$; a_k is encrypted using the key K_{sd} shared between s and d. The integers $x_1, \dots, x_l$ form a sequence of siblings of the vertices on the tree path from $h(b_k)$ up to H (Fig. 1), necessary to verify that $h(b_k)$ is a leaf of the tree, i.e., belongs to flow H.

PKT verification. To verify that a PKT $(s, d, H, b_k, f_k = [x_1, \dots, x_l], e_k, M)$ belongs to flow H, an intermediate node checks whether $h(\dots h(h(h(b_k)||x_1)||x_2)|| \dots x_l) = H$, i.e., if $h(b_k)$ is a leaf of the Merkle tree with root H. The $h(\dots h(h(b_k||x_1)||x_2)|| \dots x_l)$ is a shorthand notation, in practice the order of concatenations depends on the position of b_k in the Merkle tree. If the above check is successful, the PKT

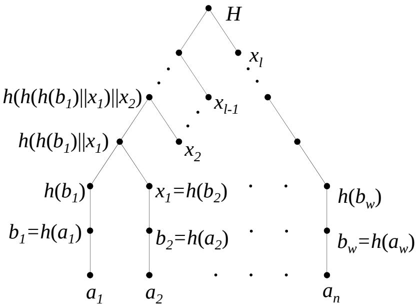


Fig. 1. **The Castor's Merkle tree scheme.** The tree construction starts by generating the sequence $a_1, \ldots, a_w$ of random numbers, which are later used to authenticate the ACKs. These numbers are then hashed to generate another sequence $b_k = h(a_k)$. The b_k sequence in turn is hashed to form the leaves of the Merkle tree. The value at each node in the tree computed by hashing the concatenation ($||$) of its children. Assume a PKT containing b_1 and the vector $H, [x_1, \ldots, x_l]$. By checking that $h(\ldots h(h(b_k||x_1)||x_2)|| \ldots x_l) = H$ the receiving node can verify that the PKT belongs to a flow identified by H.

is forwarded and b_k is stored. Otherwise, the PKT is dropped. Note that unforgeabilty of b_k/f_k follows from the hardness of inverting the hash function h.

PKT verification at destination. In addition to the Merkle tree test, the destination performs additional verification of the PKT. First, it checks whether $b_k = h(D_{K_{sd}}((e_k)))$. Then, it checks the integrity of the payload M. If all tests are successful, d accepts the PKT, and sends the corresponding ACK a_k to the neighbor that delivered the PKT. Otherwise, the PKT is dropped. Note that only the destination is able to generate a correct ACK without breaking the encryption of e_k or finding a pre-image of b_k under h.

ACK verification. Upon receiving and ACK a_k, a node computes $h(a_k)$ and checks whether it corresponds to any stored b_k. If yes, the ACK is accepted and rebroadcasted, and the routing state of the corresponding flow H is appropriately updated. Otherwise, the ACK is ignored.

Flow restarts. When the source exhausts the $a_1, \ldots, a_w$ and $b_1, \ldots, b_w$ sets for a given flow, it has to generate them anew along with the corresponding Merkle tree. This effectively starts a new flow. The in-network state for the old flow can no longer be used for routing and new state needs to be established. In practice, with a high enough value of w, the amortized bandwidth cost of reestablishing the in-network state is negligible, especially when compared to the bandwidth cost of keeping the flow's routes up-to-date under mobility.

The bandwidth overhead also depends on w in another way. The vector $[H, x_1, \ldots, x_l]$ has length $l+1$, where l is the height of the Merkle tree. The height of the Merkle tree is $\lceil \log_2 n \rceil$, thus the per-PKT overhead is logarithmically proportional to n.

In the public-key scheme presented next, the source can generate new a_is and b_is on the fly, hence flow restarts are not an issue.

2) Public key scheme:

PKT generation. Assume a flow from the source s to destination d. The source generates a public/private key pair H, H^{-1} used for generating existentially unforgeable digital signatures. The public key becomes the flow id, the private key is kept secret. For the kth PKT the ACK authenticator a_k is a freshly generated random number (a nonce); the PKT id b_k is set to $h(a_k)$ where h is a cryptographic hash function; the flow authenticator $f_k = Sig_{H^{-1}}(b_k)$ is a digital signature of b_k; and $e_k = E_K(a_k)$ is an encryption of a_k either with the symmetric key K_{sd} shared between s and d or the public key K_d of the destination.

PKT verification. To verify that a PKT $(s, d, H, b_k, f_k, e_k, M)$ belongs to flow H, an intermediate node simply verifies that f_k is a digital signature of b_k with key H. If successful, the PKT is forwarded and b_k is stored. Otherwise, the PKT is dropped. The adversary cannot forge a b_k/f_k pair without knowing the secret private key H^{-1}, or breaking the digital signature scheme, both of which are infeasible.

PKT verification at destination. The destination performs additional verification of the PKT. First, it checks whether $b_k = h(K_{sd}\{e_k\})$, using the key K_{sd} shared with the source s. Then, it checks the integrity of the payload M. If all tests are successful, d accepts the PKT, and sends the corresponding ACK a_k to the the neighbor who delivered the PKT. Otherwise, the PKT is dropped. Note that only the destination is able to generate a correct ACK without breaking the encryption of e_k or finding a pre-image of b_k under h, both of infeasible for the adversary.

ACK verification. Upon receiving and ACK a_k, a node computes $h(a_k)$ and checks whether it corresponds to any stored b_k. If yes, the ACK is accepted and rebroadcasted, and the routing state of the corresponding flow H is appropriately updated. Otherwise, the ACK is ignored.

C. PKT forwarding

Basic forwarding. For every neighbor $j = 1 \ldots n$ and for every encountered flow H, a node i stores a *reliability estimator* $s_{H,j} \in [0, 1]$. Consider what happens when i either 1) receives a PKT, and verifies that it belongs to some flow H (§V-B) or 2) i is the source of the PKT. First, i attempts to forward the PKT to the most reliable neighbor, according to the values of all the reliability estimators for the flow H. If no neighbor is deemed reliable, the PKT is broadcasted to all the neighbors in search for more reliable routes. Immediately after the PKT is sent, i starts a timer T_{H,b_k}, which times out after T_{ACK} if the corresponding ACK is not received.

The decision to unicast or broadcast is probabilistic. Let $p_{max} = \max_{j=1 \ldots n} s_{H,j}$ be the value of the highest reliability estimator for the flow H among all the neighbors $j = 1 \ldots n$ of node i. The probability that the PKT is broadcasted is $e^{-\gamma p_{max}}$, otherwise it is unicasted to the next hop with the highest reliability estimator. Ties are broken by choosing uniformly at random. The $\gamma > 0$ parameter allows for controlling the bandwidth investment in route discovery depending on the desired packet delivery rates (PDR).

Duplicate PKTs. If a node receives a PKT that it has received before, this PKT is not forwarded again. However, if an ACK corresponding to this PKT was received, the node rebroadcasts the ACK. If an intermediate node receives a PKT with b_k identical to some previously seen PKT, but with a

```
initial values
    $auth_{b_k} \leftarrow$ null, $hop_{b_k} \leftarrow$ null, $acked_{b_k} \leftarrow \emptyset$, $\alpha^{a/f}_{H,j} \leftarrow 0$, $\beta^{a/f}_{H,j} \leftarrow 1$

on receive $PKT(s, d, H, b_k, e_k, f_k = [x_1, \ldots, x_l], M)$ from $j$
    if $h(h(h(b_k||x_1)||x_2)||\ldots x_l)! = H$ or $T_{H,b_k}.expired$ then return
    if $i == d$ and $auth_{b_k} ==$ null and $M$ not corrupted then
        $a'_k \leftarrow D_{K_{sd}}(e_k)$
        if $b_k == h(a'_k)$ then
            $auth_{b_k} \leftarrow a'_k$
    if $i! = d$ and $auth_{b_k} ==$ null then
        $j_{max} \leftarrow \arg\max_{m=1 \ldots n} s_{H,m}$
        $p_{max} \leftarrow \max_{m=1 \ldots n} s_{H,m}$
        trigger timeout $T_{H,b_k}$ in $T_{ACK}$
        if $rand([0, 1]) < e^{-\gamma p_{max}}$ then
            $hop_{b_k} \leftarrow all$
            broadcast $PKT(s, d, H, b_k, e_k, f_k, M)$
        else
            $hop_{b_k} \leftarrow j_{max}$
            send $PKT(s, d, H, b_k, e_k, f_k, M)$ to $j_{max}$
    if $auth_{b_k}! =$ null then
        send $ACK(auth_{b_k})$ to $j$

on receive $ACK(a_k)$ from $j$
    $b_k \leftarrow h(a_k)$
    if $hop_{b_k} ==$ null or $T_{H,b_k}.expired$ or $j \in acked_{b_k}$ or $(hop_{b_k}! = all$
    and $hop_{b_k}! = j)$ then return
    $acked_{b_k} = acked_{b_k} \cup \{j\}$
    if $auth_{b_k} ==$ null then
        $auth_{b_k} \leftarrow a_k$
        broadcast $ACK(a_k)$
        $s^a_{H,j} \uparrow$, $s^f_{H,j} \uparrow$
    else
        $s^a_{H,j} \uparrow$

on $T_{H,b_k}$ timeout
    if $hop_{b_k}! = all$ then
        $s^a_{H,hop_{b_k}} \downarrow$, $s^f_{H,hop_{b_k}} \downarrow$
```

Fig. 2. **Protocol outline.** Castor at node i with neighbors $j = 1 \ldots n$. We have excluded the PKT duplicate checking mechanism and flood rate-limiting for brevity. The T_{H,b_k} is the timeout timer fired when waiting for the ACK. The duration of the timeout is T_{ACK}, after which $T_{H,b_k}.expired$ becomes true; $auth_{b_k}$ stores the ACK for PKT b_k, if the ACK was received; hop_{b_k} stores the neighbor to which PKT b_k was sent; in particular **null** if PKT b_k was not received; $acked_{b_k}$ stores the neighbors that sent an ACK for PKT b_k.

different payload or encrypted ACK authenticator e_k, this PKT is forwarded further. This is because an intermediate node cannot tell which of the PKTs with the same authenticator are incorrect or forged. We explain this in more detail in §VI-B. The T_{H,b_k} timer is not restarted on PKT duplicates.

D. Updating the reliability estimators

Reliability estimators. The reliability estimator $s_{H,j}$ is an arithmetic average of two reliability estimators $s^a_{H,j}$ and $s^f_{H,j}$. The $s^a_{H,j}$ estimator is updated more frequently than $s^f_{H,j}$ as we explain next. The "a" stands for "all ACKs" and "f" stands for "first ACK".

Both reliability estimators are exponential averages of packet delivery rates. More precisely, let $\alpha^a_{H,j}$ be the running exponential average of successful deliveries and $\beta^a_{H,j}$ the running exponential average of failed deliveries; then $s^a_{H,j} = \frac{\alpha^a_{H,j}}{\alpha^a_{H,j}+\beta^a_{H,j}}$. Upon a failure, the updates are: $\alpha^a_{H,j} \leftarrow \delta\alpha^a_{H,j}$ and $\beta^a_{H,j} \leftarrow \delta\beta^a_{H,j} + 1$. We denote this negative update as $s^a_{H,j} \downarrow$. Upon success, the the reliability estimator is positively ($s^a_{H,j} \uparrow$) updated: $\alpha^a_{H,j} \leftarrow \delta\alpha^a_{H,j} + 1$ and $\beta^a_{H,j} \leftarrow \delta\beta^a_{H,j}$. Initially, $\alpha^a_{H,j} = 0$ and $\beta^a_{H,j} = 1$. Updating of $s^f_{H,j}$ is analogous. The $0 < \delta < 1$ parameter controls how fast Castor adapts; the lower the value the faster the adaptation.

ACK timeout. Consider the case when T_{H,b_k} times out before the corresponding $ACK(a_k)$ is received. The update of the estimators then depends on whether the corresponding PKT was broadcasted or unicasted to some neighbor j. In the former case, no reliability estimators are changed. In the latter case, both $s^a_{H,j}$ and $s^f_{H,j}$ are decreased.

ACK reception. Consider the case when node i receives a valid ACK(a_k) from node j before the T_{H,b_k} timeout. If the corresponding PKT was unicasted and j was the next hop, both estimators $s^a_{H,j}$ and $s^f_{H,j}$ are increased, and the ACK is rebroadcasted, otherwise the ACK is ignored (e.g. when coming from a neighbor that is not j)

If the PKT was broadcasted, the behavior depends on whether it is the first ACK that was received, or not. In the former case, both estimators $s^a_{H,j}$ and $s^f_{H,j}$ are increased, and the ACK is rebroadcasted. Otherwise, only $s^a_{H,j}$ is increased and the ACK is not rebroadcasted.

For both broadcasts and unicasts, only one ACK is accepted per neighbor and per PKT id b_k; the subsequent ACKs are ignored.

Rationale behind the dual reliability estimators. Keeping track of the first ACKs with $s^f_{H,j}$ is a performance optimization. The primary routing metric in Castor is the packet delivery reliability, but $s^f_{H,j}$ gives preference to lower round-trip routes, which are typically shorter and consume less bandwidth. A similar method has been used with success in ARAN [8] and SRP [3]. On the other hand, using the second estimator, $s^a_{H,j}$, to keep track of all ACKs allows Castor to obtain more routing information with one broadcasted PKT, leading to faster convergence under attacks and when exploring routes.

E. Flood rate-limiting

To protect the system from Denial-of-Service attacks exploiting the PKT flooding (§VI) Castor uses a PKT broadcast rate-limiting mechanism. The mechanism takes advantage of the fact that messages are neighbor-to-neighbor authenticated. For each neighbor $j = 1 \ldots k$ the current allowed broadcast rate r_j is kept. The rates are initially set to one per second. When the broadcast is successful (i.e., when the ACK is received) the allowed rate is multiplied $\alpha = 2$, otherwise it is multiplied by $\beta = 0.5$. The rate values are constrained to the $[r_{min} = 1/10s, r_{max} = 100/s]$ range.

The r_j rate limit is enforced by maintaining a size 3 leaky bucket rate-limiter for each neighbor, which operates as follows. A leaky-bucket is a FIFO queue with a fixed maximum size (three, in our case). The packets can arrive at the queue at any rate but the queue transmits the packets at a constant rate r. When the queue size is already at its maximum the packet is dropped, otherwise it is enqueued. The rate limit can be adjusted by changing r.

The rate-limiting mechanism affects only the PKT broadcasts, PKT unicasts and all other messages are unaffected. We have found that this simple approach provides a strong defense against the flooding-based DoS attacks. We confirm that experimentally in §VII-E.

F. Protocol state lifecycle

Castor keeps two types of state: per-b_k and per-flow. The per-b_k state is allocated when a PKT with some b_k is received and there is no state for that b_k yet. The b_k state is deallocated

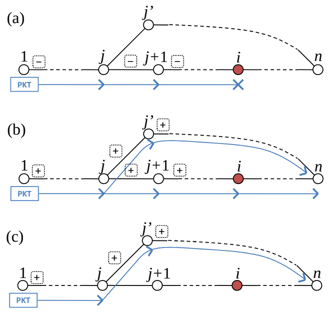


Fig. 3. **Dropping attack defence.** Reliability estimator increase indicated by "+", decrease by "-".

after T_{ACK}, the ACK timeout. The per-b_k state is used to keep track of which next hop was chosen for the b_k packets, whether an ACK arrived and what was its a_k authenticator and to track what packets with the same b_k have been sent (for duplicate PKT handling). The per-flow state is allocated the first time a packet from the given flow H is received and deallocated after $T_{collect}$ period of flow inactivity. The per-flow state contains one reliability estimator for each neighbor.

The biggest component of the memory overhead is the per-b_k state. If we assume a 11Mbps medium entirely saturated with 512-byte packets, a 3 second ACK timeout and a (generous) 128 bytes of state per packet then the upper bound on memory usage is 1056 kilobytes. Note that the same upper bound holds irrespective of the number of flows. The memory consumption is only dependent of the number of packets per second the medium can transmit.

VI. Security analysis

We first show that Castor is resilient to general attacks relevant to any routing protocol, and then do the same for Castor-specific attacks. For presentation clarity, the discussion in this section assumes a static network. In the evaluation section (§VII) we demonstrate that Castor's security properties also hold under mobility.

A. General Attacks

Packet dropping. An adversarial node drops all (*blackhole attack*) or some (*grayhole attack*) of the packets it is expected to forward. The fraction of packets a grayhole drops can vary over time and dropping can be selective, affecting only specific types of packets.

Consider a packet flow with id H from some correct source 1 to some correct destination n. Assume that the packets are forwarded by nodes $1, 2, \ldots, n-1, n$, and that one of the nodes, say i, is adversarial (Fig. 3).

If i drops a PKT, n does not receive it and does not respond with an ACK. Our acknowledgment mechanism guarantees that n is the only node able to generate an ACK for the PKT. On PKT loss (Fig. 3(a)), every node $j = 1, \ldots, i-1$ preceding i on the route times out waiting for the ACK and it decreases the reliability estimator $s_{H,j+1}$ for its successor on the route. The more aggressively i drops, the lower the estimators become. One of the following eventually happens: j broadcasts the packet to all of its neighbors (Fig. 3(b)) or the reliability estimator $s_{H,j'}$ of some neighbor $j' \neq j+1$ of j exceeds $s_{H,j+1}$, and j forwards subsequent packets to j' (Fig. 3(c)). In the case (b), some neighbors of j succeed in delivering the PKT, and respond with a correct ACK, j increases their reliability estimators and new routes are established. Eventually, after another packed drop (a) j re-routes to j', *away* from the source of unreliability. This is the fundamental mechanism through which Castor removes lossy nodes from the routes.

The protocol behavior is similar if the adversarial node i forwards PKTs, but drops ACKs: Nodes $j = 1, \ldots, i-1$ timeout waiting for the ACK and the same mechanism ensures that i is removed from the routes. The only difference to the PKT dropping is that the successors of i on the route receive the ACK and increase their respective reliability estimators. In fact, this is not undesirable, since the resultant routing state updates are correct.

Jamming. In the jamming attack, adversarial nodes prevent communication in their respective ranges. Means can vary: the attack can be mounted on the physical layer or the MAC layer, continuously or intermittently and selectively. Flows ending in the jammed regions are effectively denied communication. For other flows, a jammed region appears as a cluster of black- or gray-hole nodes and Castor is able to route around them.

Wormholes and tunnels. In a tunnel attack, remote adversarial nodes use their fast links to transfer messages out-of-band, appearing to be neighbors. In the more powerful wormhole attack, the out-of-band links are used to almost instantly relay, without any modification, messages received in one location to another remote location in the network. Thus, every node at one end of the wormhole believes to be a neighbor of every node at the other end. Route discovery mechanisms optimizing for hop-count or response time (as Castor does) are attracted by such "shortcuts", and wormholes and tunnels are likely to become a part of many routes. The adversary can take advantage of that and take control over a large fraction of the traffic, which can then be maliciously dropped or corrupted.

Castor uses the tunnels and wormholes opportunistically as long as they allow the traffic to pass through. As soon as the attacker starts dropping the packets, the PKT-ACK loop is broken and Castor turns to alternative, more reliable neighbors for routing, avoiding the lossy tunnels and wormholes. We demonstrate this property experimentally in §VII-B.

Rushing attack. Even without fast out-of-band links, the adversarial nodes can attempt to place themselves on the routes. In [21], nodes forward broadcasted PKTs as soon as possible by exploiting the MAC layer vulnerabilities. From our perspective, this attack is a weaker variant of a wormhole/tunnel attack: When the rushing nodes start dropping, they will be routed around.

Sybil attack. In a Sybil attack, a single adversarial node appears as multiple nodes to its neighbors, using the cryptographic material of other compromised nodes. As reliability estimators are kept on a per-neighbor basis, routing around a Sybil node is harder: its neighbors have to decrease their reliability estimator for each of the identities of the Sybil node. The Sybil attack is not very different from the wormhole attack. In a sense, the final outcome is identical, a node gains a number of false neighbors. These can potentially become droppers and when they do they are detected and routed around. In our performance evaluation (§VII), we focus on the most severe attack in this class of neighborhood attacks, the wormhole attack.

B. *Castor-specific Attacks*

Importance of flow isolation. Castor maintains reliability estimators per-flow, not per-destination, uses flow authenticators to identify the flows and cryptographically binds the PKTs and ACKs. This ensures that 1) in-network state for each of the flows is logically isolated and 2) only the messages originating from the source or the destination can influence the flow's state. Without this, an attacker could generate false PKT-ACK pairs for any chosen flow and maliciously modify the routing state on all the nodes that the false PKTs and ACKs traverse. This could be used, for example, to prevent PKT delivery to a legitimate destination by re-routing the traffic to an adversary-controlled node.

Message corruption and forgery. The adversary can attempt to corrupt any field of ACKs or PKTs. If the payload M of a PKT is modified, the data integrity verification fails at the destination and an ACK is not sent back to the source for that packet. Thus, any node corrupting the payload appears to its neighbors as a packet dropper, and it is routed around.

Replay attacks. As explained in §V-B, the adversary cannot successfully forge flow authenticators in PKTs or ACK authenticators. However it can be replay them. Several variants of replay attacks are possible. The objective is to influance state corresponding to legitimate active flows and attempt to reroute the PKTs in order to discard or corrupt them.

First, an adversarial node on a route can replay a forwarded PKT multiple times. Forwarding the PKT to the same neighbor has no effect, as correct nodes forward a given PKT only once. If the PKT is forwarded to different neighbors, all of them try to route it towards the destination. The resulting routing state updates are correct and do not negatively influence the network performance.

Second, the adversary could replay a PKT in another location of the network. As is the case with the local replay attack, such PKTs are routed to the destination, creating correct routing state, again, without any impact on the performance.

The adversary could, however, modify the PKT parts that intermediate nodes cannot recognize as invalid: $E_{K_{sd}}(a_k)$ and M. The Castor nodes forward every distinct copy of the PKT even if the flow authenticators are identical. This ensures the correct copy of the PKT will get through despite the nodes also receiving the malformed clones.

Considering ACK replays, observe that a given ACK can be used to increase an estimator at node j for some neighbor k at most once. Hence, it is not possible to artificially increase an estimator by repeatedly replaying an ACK from one neighbor to another. Each correct node also ignores ACKs that correspond to PKTs it never forwarded. In addition, if a PKT was unicasted to some neighbor j, the corresponding ACK from any node other than j is ignored.

Finally, the adversary can, using its fast communication links, "reenact" a correct flow in some other part of the network. This would require at least two adversarial nodes: one node i replaying PKTs and another node i' replaying the corresponding ACKs. This creates incorrect routing state all along the reenacted flow. But it has no effect on PDR, as long as the original flow is disjoint from the reenacted flow. If, for some reason, the original flow reaches one of the nodes that are reenacting the flow, then the incorrect routing state delays PKT delivery by forwarding towards i'. This state is then quickly corrected if those misrouted PKTs are not delivered. Overall, this attack cannot be sustained without delivering the PKTs to the correct destination in order to obtain the fresh valid ACKs that would be needed to reenact the flow. However, the delivery to the correct destination defeats the purpose of the attack.

Flooding attack. In Castor, the nodes broadcast the PKTs whenever no reliable route is known. An adversary could use the PKT rebroadcasts as an attack amplification device. A high-rate stream of PKTs could be injected, each PKT belonging to a distinct new dummy flow. This would trigger a network-wide flood for each PKT potentially causing global bandwidth starvation. All the routing protocols relying on flooding for route discovery have this vulnerability: an attacker can trigger floods at a high rate and cause Denial-of-Service. Very few of the existing protocols address that issue. Castor uses flood rate-limiting (§V-E) that limits how often a given neighbor can cause a PKT broadcast. We show experimentally (§VII-E) that this simple measure gives a high level of protection from the flooding attacks.

In Castpr, a neighbor is allowed a higher rate when the broadcasts it causes are followed by ACKs. An attacker could exploit that and set up a colluding node in the network that would be ACKing the bogus PKT stream and thus allowing a higher attack rate. However, this also means that there would be enough bandwidth left for some of the benign messages, which weakens the attack. In our experiments we found this attack variant is much more difficult to perpetrate and that it has lower impact on performance than the simple non-ACKed PKT flooding. The rate limiters could be precisely tuned to provide a very strong defence against the ACKed flooding, but we leave this for future work.

VII. Performance Evaluation

Our evaluation is carried out using the ProtoPeer (`http://protopeer.net`) message passing framework, with JiST/SWANS (`http://jist.ece.cornell.edu/`) employed for MANET modeling. Apart from Castor, we have implemented three other routing protocols: Sprout, SRP and SEAD. We use the default-setting implementation of AODV from the JiST/SWANS library. The choice of protocols covers all combinations of on-demand/proactive and distance-vector/link-state categories (Fig. 4). Our Sprout implementation uses the parameters recommended in [13], with one exception: To handle mobility, the maximum number of routes stored at a given time is set to 50. When exceeded, the lowest-ranked

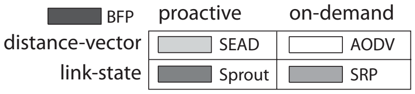


Fig. 4. **Legend.** The suite of evaluated protocols covers the whole matrix of protocol types. Castor could be classified as an on-demand protocol, but does not fall into either the distance-vector or link-state categories, since in Castor no network topology information is ever exchanged.

route is removed. The SSP [1] protocol is layered on top of SRP [3].

We use the random waypoint mobility model. Nodes send neighbor discovery beacons every $250ms$. A node is removed from the neighbor set if not heard from for $1s$. Unless otherwise stated, the measurements are averaged over 50 independent runs, 1 hour of simulated time each. Every data point in the time-involving measurements is a 10 second average. Each of the 50 runs has distinct node trajectories. The same trajectories are repeated for all the protocols. We show 90% confidence intervals. The experimental setup parameters are summarized in Table I.

TABLE I
EXPERIMENTAL SETUP

General			
plane size		3km by 3km	
nodes		100, placed uniformly at random, 10 radio neighbors on average	
MAC		802.11b at 1Mbps	
number of flows		5, source-destination disjoint	
flow rate		constant bit rate, 4 packets/s	
packet payload size		256 bytes	
Random waypoint mobility		**Castor**	
min. speed	1 m/s	γ	8
max. speed	20 m/s	δ	0.8
pause time	0s	T_{ACK}	500ms
SEAD			
periodic route update interval	5s		
SRP		**Sprout**	
α, β, δ	0.5	γ	1.25
r_s^{thr}	0	α_{pdr}	0.9
r_s^{max}	1	α_{rtt}	0.9

A. Dropping attack

We implement selective blackholes, dropping all data packets, but allowing control packets (route discovery) through. For Castor, the attacker drops unicasted PKTs, and forwards broadcasted PKT to attract more routes. Fig. 5 shows the achieved packet delivery rate (PDR), varying the fraction of adversarial nodes. Recall that we look at the network-layer PDR, i.e., there are no retransmissions to mask the packet loss.

The AODV and SEAD protocols do not make any end-to-end checks for packet loss and they are unaware of the blackholes. The performance of the two protocols is thus significantly affected. Sprout and SRP monitor route reliability, and thus significantly improve over SEAD and AODV. However, Sprout and SRP do per-route performance accounting; when the fraction of attackers is high, most of the routes contain at least one adversarial node and both protocols take longer time to converge on a blackhole-free route. In contrast, Castor keeps reliability records with higher granularity, per-link instead of per-route, and can detect and route around the attackers much faster.

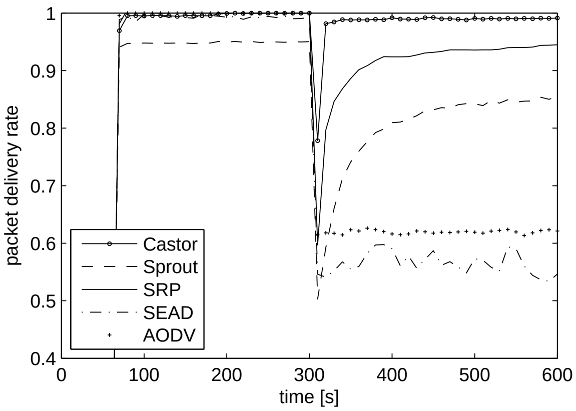


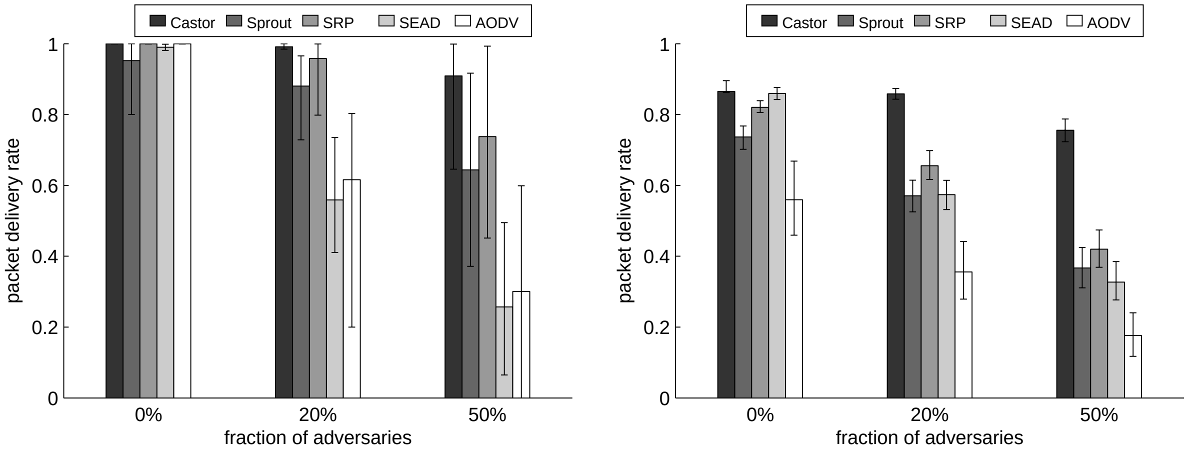
Fig. 6. **Failure recovery time under blackhole attack.** There are 100 immobile nodes. The flows start at the 1 minute mark. At the 5 minute mark 20 nodes become blackholes. The results are averaged over 50 independent runs, 5 simultaneous flows in each run. By keeping track of reliability per-link, Castor quickly locates the blackholes and recovers within 30s. This is in contrast to SRP and Sprout, which keep reliability records per-route and converge slower or might entirely fail to find blackhole-free routes.

Failure recovery time. The benefits of storing reliability information per-link rather than per-route are clearly demonstrated by the experiment on Fig. 6. Castor recovers in under 30s, much faster than SRP or Sprout. This even more clearly shows that storing reliability information per-link is superior to storing it per-route. This enables Castor to more rapidly pinpoint the location of the blackholes and reach full recovery in under 30s. In contrast, the other protocols converge slower since only the sources are engaged in evaluating the route reliability and not the whole network as in Castor. The number of possible routes to test for loss in Sprout and SRP is combinatorially larger than the number of links to test in Castor.

Even after 30 minutes, Sprout and SRP fail to find reliable routes for some of the flows, which is reflected in the 50-run averages. AODV and SEAD, not surprisingly, do not recover from the attack.

Mobility. While mobility degrades the performance of all protocols, Castor is the least affected (Fig. 5(b)): Nodes observe the topology changes locally and most of the time they are able to select an alternative next hop on the spot or perform a local flood to repair the route. In contrast, the other protocols must either: (i) wait for the new topology information to propagate through the network (Sprout and SEAD) or (ii) wait for the on-demand route (re)discovery to finish (AODV, SRP). In addition, the newly discovered routes must be "evaluated" (Sprout, SRP) for the presence of the black holes.

Bandwidth utilization. Bandwidth utilization for the performed experiments is shown in Fig. 7. Castor is unique among the five protocols merging the routing and route discovery phases into one. This comes at the cost of including the full



(a) No mobility

(b) With mobility

Fig. 5. **Blackhole attack resilience.** We vary the fraction of blackholes in the system with and without mobility. The error bars indicate 90% confidence intervals. Both SEAD and AODV do not have any defenses against the blackhole attacks and suffer the biggest performance drop. Sprout and SRP track the reliability on the per-route basis and when there are more blackholes, most of the routes are likely to contain at least one adversarial node and the two protocols struggle to find an adversary-free path. Castor tracks the reliability per-link and is able to locate and route around the adversarial nodes more accurately.

256 byte data payload in the flooded PKTs. However, often Castor responds to changing network conditions by simple re-routing or limited flooding (§V-C), which results in amortized bandwidth cost comparable to the other protocols.

The two proactive protocols, Sprout and SEAD, require additional bandwidth for propagating the network topology information. Under mobility, even in the benign case, Sprout uses a substantial amount of bandwidth for link-state updates.

Grayhole attacks. We have so far evaluated the protocols under the black hole attack, where the attacker drops all the data packets. To make the attack more challenging to detect, the attacker could mount a *grayhole attack* in which only a fraction of the data packets are affected. In the next experiment the settings is identical to the blackhole setup, except that now the attacker drops 50% of the packets uniformly at random. The results are presented on Fig. 8.

The packet delivery rate of Castor, Sprout and SRP are almost the same as in the blackhole case (Fig. 5). However, the protocols take more time to identify the attacker nodes and route around them, when compared to Fig. 6. Unsurprisingly, AODV and SEAD suffer a much smaller performance drop, since the attacker affects twice fewer packets.

B. Wormhole attack

We set up a wormhole with three exit points. The points form an equilateral triangle, each pair of points separated by 1000m. The wormhole is implemented at the radio layer. Initially, the wormhole forwards all the packets. At the 5 minute mark, the wormhole stops retransmitting any data traffic, but still keeps retransmitting the control traffic and broadcasted PKTs. Out of all the wormhole behaviors the we tried, this one had the most severe impact on the performance of all the protocols. We measure how the PDR changes in response to the attack (Fig. 9).

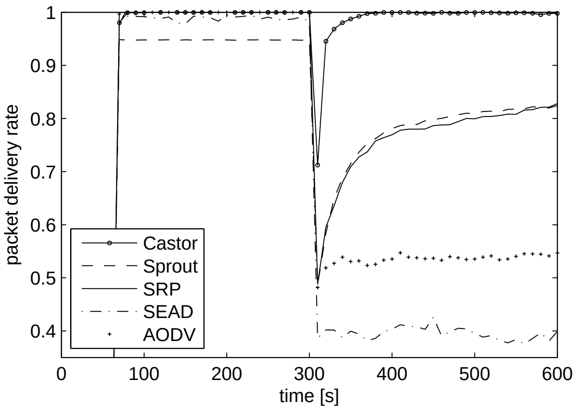
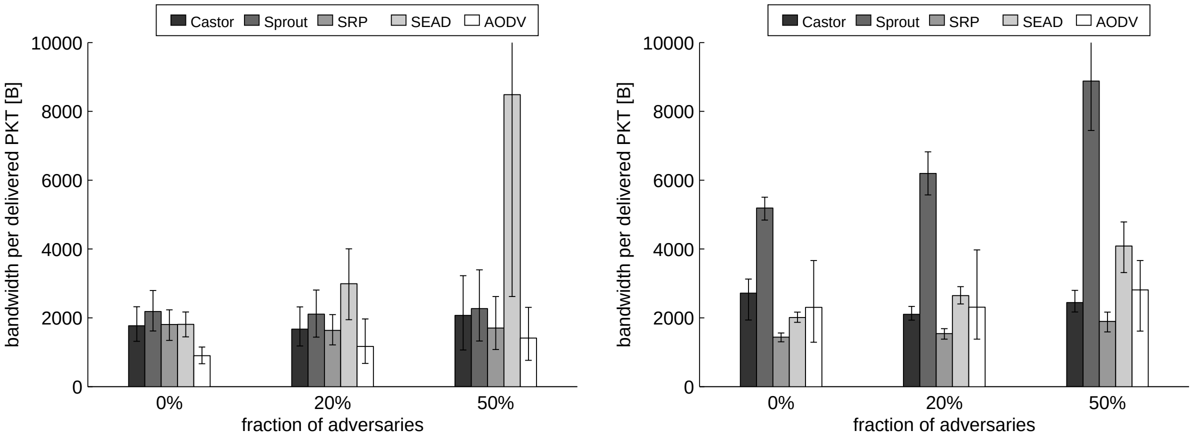


Fig. 9. **Wormhole attack resilience.** There are a 100 immobile nodes, a triple exit point wormhole is present. The wormhole is initially passive, but at the 5 minute mark starts dropping all the data traffic. The results are averaged over 50 independent runs, 5 simultaneous flows each. Castor is the only protocol of the five that fully recovers from the wormhole attack.

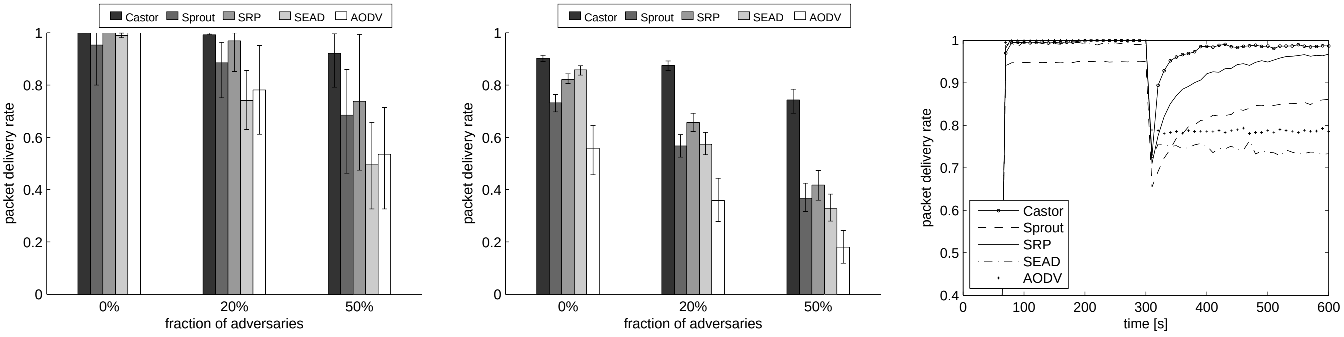
Initially, all protocols are attracted to the shorter routes the wormhole offers. After the wormhole stops transmitting data traffic, AODV and SEAD continue to route through the wormhole, because the control traffic goes through. As a result, they suffer from significant packet loss, which is not zero only because some source/destination are located close enough not to be attracted to the wormhole routes. SRP and Sprout gradually switch to routes without any wormhole links. However, because only the source evaluates the routes, convergence is much slower than with Castor. In fact, for some flows SRP and Sprout cannot find an adversary-free route within the simulation time, and thus do not fully recover. Note, that Sprout has not been designed to defend against the wormhole attacks [13] and instead the authors recommend to rely on solutions such as TrueLink [22]. We did not simulate TrueLink. Castor recovers from wormholes completely without any additional wormhole



(a)

(b)

Fig. 7. **Bandwidth utilization under the blackhole attack.** Left: no mobility. Right: mobility. The experimental setup identical to Fig. 5. Even though Castor floods the network with PKTs that include the full data payload, the amortized bandwidth cost is comparable to the other protocols. The proactive protocols consume additional bandwidth while exchanging the network topology updates.



(a) No mobility

(b) With mobility

(c) No mobility, failure recovery time

Fig. 8. **Grayhole attack resilience.** We vary the fraction of grayholes in the system with and without mobility. All protocols except AODV and SEAD defend against the grayholes just as well as the blackholes (Fig. 5). However, the time required to detect the attacker is longer.

defense mechanisms and their overhead.

Other attacks. We did not evaluate tunnel, rushing or Sybil attacks, as from our perspective they are very similar in nature, but weaker than the wormhole attack. We also omit the evaluation of the replay attacks; as show in §VI-B, they do not pose a significant threat.

C. Performance under mobility

To test how node mobility influences the protocols' ability to detect the adversary, we set the number of blackholes to 20%, and vary the node pause time in the random waypoint model measuring the PDR (Fig. 10).

Castor's local failure detection and repair is able to rapidly reroute the PKTs, when nodes go beyond the radio range or the new neighbor turns out to be a black hole. Sprout and SRP need to route more PKTs to determine which routes are reliable and often this process is slower than the rate of change in the topology. AODV and SEAD display constant performance, confirming that its not the mobilty that affects them, but rather the attack.

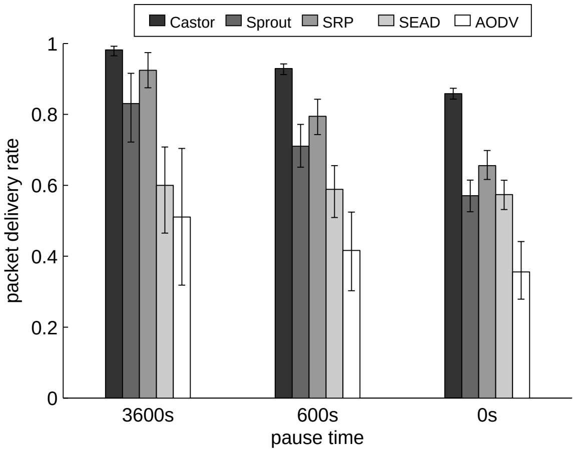


Fig. 10. **Performance under mobility.** There are a 100 nodes, out of which 20 are blackholes. We vary the pause time. Castor is fast both at detecting the blackholes and reacting to topology changes, while the other protocols either do not have countermeasures against the blackholes or their detection processes are slower than the rate of change of the network topology.

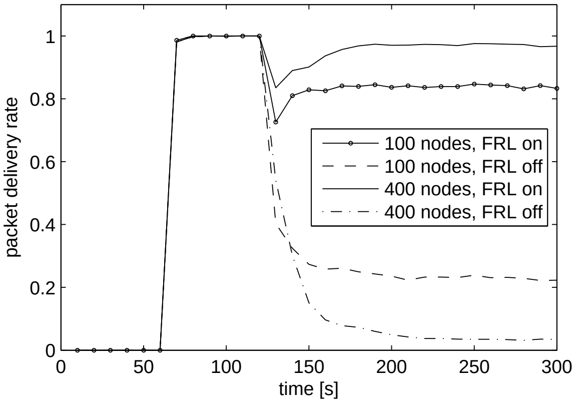


Fig. 12. **Flooding attack resilience.**The attack begins at the 120s mark. The five 4pkt/s flows begin at 60s. The curves are averages over 50 independent randomly seeded runs.

D. Scalability

We next measure the performance of the protocols for different network sizes, keeping node density constant; 20% of the nodes are blackholes under zero pause time mobility. The results are shown in Fig. 11.

The routing paths get longer with the increasing network size and finding an adversary-free path becomes more challenging. The longer paths are also more likely to break due to mobility. Under these conditions Castor still outperforms the other protocols and maintains a 60% packet delivery rate on a 6km by 6km plane with 400 mobile nodes and 80 blackholes.

As the network size increases, the paths get longer and more damage is caused by mobility, making Castor resort to PKT flooding more often, which the bandwidth measurements confirm. The bandwidth utilization of the proactive protocols (SEAD and Sprout) significantly increases. With 400 nodes, Sprout experiences a congestion collapse as the network is overflooded with link-state updates.

E. Flooding attack resilience

Without the flood rate-limiting (FRL) mechanism (§V-E) Castor is vulnerable to the flooding attack (§VI-B). In what follows, we experimentally demonstrate the ability of FRL to thwart such an attack.

An attacker controls a single node, which starts 200 new dummy flows per second by broadcasting PKTs, each containing a new unique flow authenticator. With FRL inactive, the attack prevents a large fraction of the traffic from passing through (Fig. 12). With FRL active, however, the nodes quickly detect and contain the attacker and reduce the effect of the attack. Complete recovery may not be possible; in some cases the source or destination reside in the neighborhood of the the rapidly broadcasting attacker, which effectively prevents local communication. In a larger network, the routing paths are longer on average and it is easier for the malicious PKT flood to disrupt them. However, when FRL is active in the larger network, the attack is contained to a smaller area relative to the whole plane size, thus the performance decrease is smaller.

F. Evaluation summary

Resilience to a wide spectrum of attacks. Unlike the other evaluated protocols, Castor does not explicitly separate the route discovery and routing phases and uses the same fault-tolerance mechanism to protect both of them. The cryptographic scheme used in Castor allows each intermediate node receiving an acknowledgment to securely verify that the acknowledgment originated at the destination. Each node locally attempts to improve its routing decisions with a single goal in mind: maximize the number of received correct acks. In this way, Castor's fault tolerance mechanism abstracts away from the causes of the failures, which makes the protocol resilient to a broad spectrum of attacks.

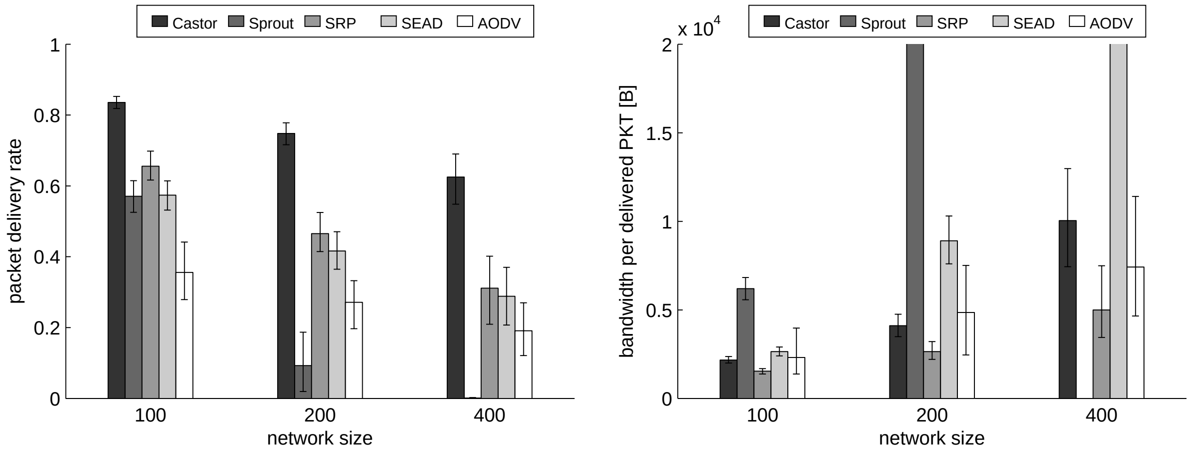
Our evaluation confirmed that Castor is resilient to blackhole and grayhole attacks even when half of the nodes are compromised. Castor is also the only protocol out of the evaluated ones that can fully and consistently recover from the wormhole attacks. As we have argued in §VI, any attacks that have a local effect and cause loss (e.g. jamming) are eventually routed around by Castor. In addition, because of the autonomy of the nodes and the isolation of the flow state (§V-B), Castor is immune to collusion, rushing and replay attacks.

Fast recovery. Sprout and SRP are both source routing protocols and build a reliability model of the network at the sources. The model is constantly updated based on the arriving or timing out acks and the sources adapt to changing loss and delay conditions. In contrast, in Castor the reliability model is distributed; each node locally keeps a performance record of its neighbors. The reliability information is kept exactly where it is needed to make the next hop decisions and can be rapidly updated in reaction to local events. In the other protocols, the information about the changes in the network take longer to reach the route planner (the source), thus substantially slowing down the recovery time compared to Castor. Moreover, Castor keeps the performance records per-link and not per-route, which allows it to pinpoint the failures and the attackers more accurately.

Scalable under mobility. As measured in our evaluation, the proactive protocols (SEAD and Sprout) do not scale. As the network grows, more link-state and routing table information must be exchanged; soon this leaves very little room in the medium for the data traffic, especially under moderately high mobility. In contrast, Castor is an on-demand protocol, which already lowers the bandwidth consumption, but what is more, Castor floods only conditionally and the floods tend to be limited to the part of the network that has been damaged by mobility. As we have shown, this allows the protocol to scale even under a zero pause time mobility and significant (20%) presence of the adversary.

VIII. Discussion

Potential for cross-layer design. In our evaluation we measured the raw packet delivery rate of Castor, without considering packet retransmissions to mask the failures. It is an open question how retransmissions would interact with Castor's fault-tolerance mechanisms and if this would not make the



(a)

(b)

Fig. 11. **Scalability.** We increase the number of nodes and the plane size, while maintaining the same node density. There are 20% of blackholes under constant mobility (pause time is zero). The packet delivery rate of all the protocols drops as the network scales up and the longer routing paths break more frequently under mobility. Castor outperforms the other protocols The proactive protocols (SEAD, Sprout) do not scale. With 400 nodes, Sprout suffers from congestion collapse caused by the link-state updates saturating the available bandwidth.

protocol more vulnerable to attacks. Moreover, the protocol uses only two types of messages: PKTs and ACKs, this opens up the possibility of cross-layer integration with the flow control protocols such as TCP or ATP [23], which use exactly the same messaging pattern as Castor.

Optimizing the dynamics. The dynamics of route discovery and failure detection are driven in Castor by two constants: γ and δ. The γ parameter controls the process of deciding whether the current best next hop has an acceptable packet delivery rate or whether the PKT should be broadcast in search for better routes. The δ parameter controls the rate of change of the reliability estimators and the protocol's sensitivity to failures. The two parameters have a significant impact on Castor's performance and attack resilience. We tuned these values experimentally but a more formal understanding of Castor's dynamics is needed to determine the optimal operating point.

Improving the adaptivity. The ACK timeout, T_{ACK}, was fixed to 500 ms for all the scenarios in the evaluation. However, an adaptive timeout, based on the current round trip time measurements for the given destination, could potentially improve the performance by allowing the protocol to react to loss as soon as it happens.

IX. Conclusions

We proposed Castor, a novel secure communication protocol for ad hoc networks. Despite the very simple PKT-ACK messaging, the protocol is more resilient to attacks than any previously proposed secure communication protocols, as demonstrated by the extensive comparative evaluation. Moreover, Castor abstracts away from the causes of the failure; any network event leading to packet loss, benign or adversarial, is detected and the routes continuously adapt to maintain high packet delivery rate.

Each Castor node locally keeps a performance record of its neighbors. The reliability information is kept exactly where it is needed to make the next hop decisions and can be rapidly updated in response to local events, which as we confirmed in the measurements, allows for fast failure recovery and fast adaptation under mobility. All that is achieved while relying on weak, and thus more practical, trust assumptions.

Several interesting open issues remain. Among them: How would Castor interact with a reliable transfer protocol? Can the PKTs and ACKs be taken advantage of for cross-layer design? How to tune the parameters of Castor, notably the ones controlling the broadcast vs. unicast behavior, to achieve the right balance between route exploration vs. exploitation? Could one extend the reliability estimators to measure both loss and delay? Finally, how do the potential solutions to the above problems affect the protocol security?

References

[1] P. Papadimitratos and Z. J. Haas, "Secure Data Communication in Mobile Ad Hoc Networks," *IEEE JSAC*, vol. 24, no. 2, 2006.

[2] W. Galuba, P. Papadimitratos, M. Poturalski, K. Aberer, Z. Despotovic, and W. Kellerer, "Castor: Scalable secure routing for ad-hoc networks," 2009.

[3] P. Papadimitratos and Z. Haas, "Secure routing for mobile ad hoc networks," in *SCS CNDS'02*.

[4] Y. Hu, A. Perrig, and D. Johnson, "Ariadne: A secure on-demand routing protocol for ad hoc networks," *Wireless Networks*, vol. 11, no. 1, 2005.

[5] G. Acs, L. Buttyan, and I. Vajda, "Provably secure on-demand source routing in mobile ad hoc networks," *IEEE TMC*, vol. 5, no. 11, pp. 1533–1546, 2006.

[6] P. Papadimitratos and Z. Haas, "Secure Link State Routing for Mobile Ad Hoc Networks," in *Proceedings of the IEEE Workshop on Security and Assurance in Ad Hoc Networks*, 2003.

[7] P. Papadimitratos, Z. Haas, and J. Hubaux, "How to Specify and How to Prove Correctness of Secure Routing Protocols for MANET," in *IEEE BROADNETS'06*.

[8] K. Sanzgiri, B. Dahill, B. Levine, C. Shields, and E. Belding-Royer, "A Secure Routing Protocol for Ad-hoc Networks," in *IEEE ICNP'02*.

[9] M. Zapata, "Secure ad hoc on-demand distance vector routing," *ACM Mobile Computing and Communications Review*, vol. 6, no. 3, 2002.

[10] C. Perkins, E. Belding-Royer, S. Das *et al.*, "Ad hoc on-demand distance vector (AODV) routing," RFC 3561, 2003.

[11] Y. Hu, D. Johnson, and A. Perrig, "SEAD: Secure efficient distance vector routing for mobile wireless ad hoc networks," *Ad Hoc Networks*, vol. 1, no. 1, pp. 175–192, 2003.

[12] P. Papadimitratos and Z. Haas, "Secure message transmission in mobile ad hoc networks," *Ad Hoc Networks*, vol. 1, no. 1, pp. 193–209, 2003.

[13] J. Eriksson, M. Faloutsos, and S. Krishnamurthy, "Routing amid colluding attackers," in *IEEE ICNP'07*, 2007.

[14] B. Awerbuch, R. Curtmola, D. Holmer, C. Nita-Rotaru, and H. Rubens, "Odsbr: An on-demand secure byzantine resilient routing protocol for wireless ad hoc networks," *ACM TISSEC*, vol. 10, no. 4, 2008.

[15] L. Buttayan and J.-P. Hubaux, *Security and Cooperation in Wireless Networks*. Cambridge University Press, 2007.

[16] G. Di Caro and M. Dorigo, "AntNet: Distributed stigmergetic control for communications networks," *Journal of AI Research*, vol. 9, no. 2, pp. 317–365, 1998.

[17] S. Marwaha, C. Tham, and D. Srinivasan, "A novel routing protocol using mobile agents and reactive route discovery for ad hoc wireless networks," in *ICON'02*.

[18] O. Hussein and T. Saadawi, "Ant routing algorithm for mobile ad-hoc networks (ARAMA)," in *IEEE IPCCC'03*, pp. 281–290.

[19] A. Perrig, R. Canetti, J. Tygar, and D. Song, "The TESLA broadcast authentication protocol," *RSA CryptoBytes*, vol. 5, no. 2, pp. 2–13, 2002.

[20] P. Papadimitratos, M. Poturalski, P. Schaller, P. Lafourcade, D. Basin, S. Čapkun, and J.-P. Hubaux, "Secure neighborhood discovery: A fundamental element for mobile ad hoc networking," *IEEE Communications Magazine*, vol. Vol.46, No.2, February 2008.

[21] Y. Hu, A. Perrig, and D. Johnson, "Rushing attacks and defense in wireless ad hoc network routing protocols," in *ACM WiSe'03*.

[22] J. Eriksson, S. Krishnamurthy, and M. Faloutsos, "Truelink: A practical countermeasure to the wormhole attack in wireless networks," in *IEEE ICNP'06*.

[23] K. Sundaresan, V. Anantharaman, H. Hsieh, and A. Sivakumar, "ATP: A reliable transport protocol for ad hoc networks," *IEEE TMC*, vol. 4, no. 6, pp. 588–603, 2005.