Xcastor: Secure and Scalable Group Communication in Ad Hoc Networks

Milan Schmittner, Matthias Hollick
Secure Mobile Networking Lab, Technische Universität Darmstadt, Germany
{milan.schmittner, matthias.hollick}@seemoo.tu-darmstadt.de

***Abstract*—Mobile ad hoc networks (MANETs) are emerging as a practical technology for emergency response communication in the case a centralized infrastructure malfunctions or is not available. Using smartphones as communication devices, MANETs may be readily established among the civilian population of affected areas. Beneficiaries would be civilian first responders, which may form small collaborating groups. Communication in such groups must be reliable for disaster response to be effective. In this paper, we address the issue of reliable group communication on the network layer. We design and implement the first secure explicit multicast routing protocol called Xcastor, in which we extend the secure and scalable routing concept of the Castor unicast routing protocol towards supporting reliable communication for large numbers of small groups. By simulation, we show significant performance improvements over Castor.**

I. Motivation

Mobile ad hoc networks (MANETs) are decentralized and autonomous communication systems: participating nodes (smartphones, laptops, etc.) forward messages to one another in a multi-hop manner, thereby creating networks spanning large areas. Such networks can support freedom of speech in countries with governmental Internet restrictions [1]. In such a scenario, the network must be able to withstand DoS by a powerful nation-state adverary. Also, MANETs can act as temporal backup networks when a natural or man-made disaster has brought down the local telecommunication infrastructure [2]. In this case, civilian first responders need to help the injured and distribute resources until professional emergency response teams arrive at the scene and erect a backup infrastructure. In both applications, people will form self-organizing, collaborative groups. Efficiency and success of those groups depends on the availability and performance of a reliable multicast communication substrate.

Challenge. A core problem of MANETs is their poor *scalability*. Apart from information theoretic capacity boundaries [3], the dominating capacity-limiting factor is the control overhead of NET and MAC layer protocols [4]. Thus, protocols should minimize the amount of overall data dissemination, and keep traffic as local as possible. Other problems in MANETs relate to reliability, e. g., (a) channel error-induced packet loss, (b) unstable nodes, and (c) node mobility cause frequent topology changes which need to be addressed. *Security* is an equally important aspect: an adversary able to bring down local telecommunication infrastructure might very likely have the resources to also disrupt a backup ad hoc network. In this sense, we focus on the security goals of availability, authenticity, and integrity of the routing service.

We summarize the challenge addressed in this work as follows: we aim at *enabling secure and scalable multicast for wireless communication on resource-constrained nodes in hostile environments*. More precisely, a solution to this problem should (a) provide reliable data transfer, (b) be able to withstand denial-of-service (DoS) attacks, (c) be robust against frequent topology changes, and (d) introduce as little overhead as possible.

Contribution. We propose the first secure explicit multicast routing protocol for MANETs called "Xcastor". The design is based on Castor [5], a flow-based security-first unicast routing protocol, and the concept of explicit multicast. Xcastor introduces the concept of *per-destination subflows* which on the one hand sandboxes flow state towards each destination in the multicast group to maintain Castor's strong security properties, and on the other hand is efficient in the sense that it scales to a large number of small groups, does not introduce per-group state information or control traffic, and draws only on efficient symmetric-key cryptography. We experimentally compare Xcastor with Castor and a flooding protocol and show that Xcastor achieves better bandwidth utilization (20 %) and shorter end-to-end delivery delays (36 %) than Castor.

Outline. In Section II, we review related work. Based on the system model and background on Castor (Sections III and IV), we present our protocol design in Section V. Section VI contains implementation details. In Section VII, we present and discuss our simulation results, and we conclude the paper in Section VIII.

II. Related Work

Multicast prevents duplicate information from being transmitted over the same link (Fig. 1). To achieve this, *stateful* multicast protocols build either (single-path) spanning-trees [6] or (multi-path) meshes [7], [8]. Stateful protocols need to maintain network-wide state for every group, supporting a *few large* groups in the network. *Stateless* approaches are either based on probabilistic gossip [9] or apply the concept of explicit multicast (Xcast) [10]. Xcast protocols enumerate all destinations in the packet header and let a unicast protocol provide the routing decisions for each destination. Hence, they do not require maintenance of a separate multicast structure

This work has been funded by the LOEWE initiative Hessen, Germany within the NICER project.

978-1-5090-2185-7/16/$31.00

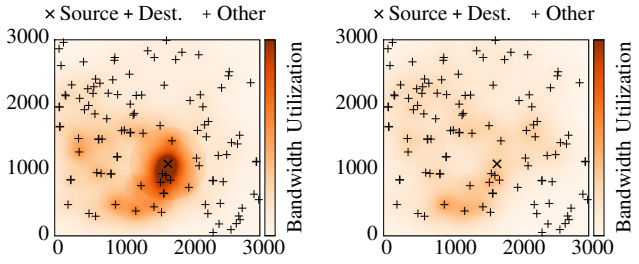

Fig. 1: Qualitative comparison of Castor (*left*) and Xcastor (*right*) addressing a group size of 10 nodes. Castor exhibits a traffic "hot spot" around the source which is eliminated by our Xcastor protocol.

and can consequently support an arbitrarily large number of groups. However, they scale poorly with group size, and are therefore often referred to as "small group multicast". Most work on Xcast for MANETs has attempted to improve scaling for larger groups by introducing group membership caches or node hierarchies [11], [12], [13].

Work on secure MANET multicast is sparse: to our knowledge, only secure tree-based multicast has been considered [14], [15]. However, the practical relevance of these proposals is questionable, as tree-based approaches lack alternate paths needed to cope with route failures common in MANETs. In contrast, numerous secure unicast protocols have been proposed, offering different levels of security and applicability: SAODV [16] and SRP [17] are vulnerable to the tunneling attack. ARAN [18] completely prevents message alteration but relies on computationally expensive public-key cryptography at every hop. Ariadne [19] successfully routes around multiple colluding malicious nodes, but requires security associations between all nodes on the path. More recently, Castor [5] offers resilience against a vast range of attacks, is highly adaptive to topology changes, and has minimal trust assumptions. It essentially satisfies our requirements in Section I for the unicast case.

III. System Model

We base our work on a network, trust, and adversary model.

Network Model. We assume a MANET with the following properties: (a) *Shared broadcast channel*. All nodes are equipped with omnidirectional antennæ so that a node's transmission can be overheard by any of its neighbors. This requires the presence of an access control mechanism at the data link layer. In this work, we focus on IEEE 802.11. (b) *Symmetric links*. The transmission range is fixed for all nodes, resulting in bidirectional links. (c) *Continuously connected nodes*. The network is sufficiently dense such that there always exists at least one (multi-hop) path connecting any two nodes (or group of nodes) that want to communicate. Otherwise, routing would not be possible. (d) *Mobility.* Nodes may freely move around as long as (c) is not violated.

Trust Model. A minimal trust model is assumed. Specifically, we assume three levels pre-established of security associations: (a) between a source s and every destination in the group d_i referred to as K_{sd_i}; (b) between all group members to protect group message integrity. (c) between neighbors, so that they can authenticate one another to protect against flooding-based DoS attacks.

Adversary Model. Our adversary model is a weaker variant of the Dolev–Yao man-in-the-middle adversary [20]. Specifically, the adversary (a) can be an authenticated member of the network taking part in the protocol execution, and might even be part of a multicast group ("insider"); (b) can eavesdrop on the communication and interfere with the protocol operation by message manipulation, forgery, and dropping; (c) can control a portion (not all) of the communication links such that there always exists an adversary-free path between any two nodes in the network (otherwise DoS attacks could not be prevented); and (d) cannot break cryptographic primitives. In this model, an adversary is able to conduct attacks including—but not limited to—route spoofing, rushing, wormhole, sybil, and black-/greyhole attacks. In this work, we are not concerned with the prevention of passive attacks (sniffing) or attacks on lower layers (MAC and PHY).

IV. Background on Castor

Our protocol builds on Castor [5], a secure and scalable unicast routing protocol for MANETs. Below, we briefly discuss Castor's core concepts.

- *Implicit route discovery*. Castor builds routes implicitly. Initially, data packets (PKTs) are flooded through the network. Acknowledgment packets (ACKs) from the destination are then used to dynamically build and adapt routes according to the distance-vector routing principle.
- *Reliability as (primary) distance metric*. Rather than using hop count as a routing metric, Castor considers the reliability of a link for transmissions towards a destination. This approach takes both accidental as well as deliberate packet loss into account. "Reliability" is defined as the past behavior of an individual neighbor, and measured as the moving average of the packet delivery ratio. The round-trip time is used as a secondary metric.
- *Independent route exploration vs. exploitation*. Nodes independently decide whether to perform route exploration (broadcast) or route exploitation (unicast) for an individual PKT. The decision is made probabilistically based on whether or not a reliable route is known.

Communication in Castor is flow-based. The PKT is a tuple $\langle s, d, H, b_k, f_k, a_k, \mathcal{P} \rangle$, where s and d are the source and destination identifiers; H is the flow identifier; b_k is the PKT identifier; f_k is the flow authenticator; a_k is the PKT authenticator; and $\mathcal{P}$ is the user payload, which may be encrypted. The entire PKT must be integrity-protected. The index k denotes the kth PKT of flow H. We denote $\text{ENC}_{K_{sd}}(\cdot)$ and $\text{DEC}_{K_{sd}}(\cdot)$ as a pair of symmetric encryption and decryption functions; K_{sd} is a shared key between s and d. $\text{H}(\cdot)$ is a cryptographic hash function. The flow ID H is the root of a Merkle hash tree, where $\text{H}(b_k)\,, k \in \{1, \cdots, 2^l\}$, are the leaves. $f_k = \langle x_1, \ldots, x_l \rangle$ contains sibling tree elements leading from b_k to H, which enables any forwarding node to verify that a PKT with b_k belongs to flow H (cf. Eq. (4)). The ACK consists of the ACK authenticator a_k, which can

Algorithm 1 Basic explicit multicast

function FORWARD(PKT($\mathcal{D}$))
  for all $\mathcal{D}_j := \{d \in \mathcal{D} : \text{NEXT\_HOP}(d) = h_j \in \mathcal{N}\}$ **do**
    if $\mathcal{D}_j \neq \emptyset$ **then**
      UNICAST PKT($\mathcal{D}_j$) to h_j
  $\mathcal{D}_\mathcal{N} := \{d \in \mathcal{D} : d \notin \bigcup \mathcal{D}_j\}$
  if $\mathcal{D}_\mathcal{N} \neq \emptyset$ **then**
    BROADCAST PKT($\mathcal{D}_\mathcal{N}$) (to $\mathcal{N}$)

be generated by the destination by decrypting a_k. If an intermediate node receives a_k, it knows that the destination must have received the corresponding PKT and can safely update its routing state.

To summarize, Castor maintains correct routing state by a scheme that satisfies two properties: (1) only the source node is able to generate PKT IDs b_ks belonging to a flow H, and (2) valid ACKs can only be received by intermediate nodes if the destination has actually received the corresponding PKT.

V. Xcastor Design

We present the design of our novel secure multicast protocol called Xcastor. We start with an unsecured toy protocol to illustrate the basic idea. Then, we introduce the concept of *per-destination subflows* which enables Xcastor to operate efficiently and securely, and explain the technical workings of the protocol in detail.

A. Explicit Multicast

In Xcast, the source specifies a list of destinations (i. e., the group members) $\mathcal{D} = \langle d_1, \ldots, d_n \rangle$ in each packet header. A node makes independent forwarding decisions for each destination d_i in $\mathcal{D}$. In our protocol, Castor makes the decision, i. e., for every destination a node decides to either unicast to a single neighbor or broadcast to all neighbors $\mathcal{N}$. The combination of Xcast and Castor results in Algorithm 1. We denote h_j as a neighbor identifier, $\mathcal{D}_j$ as the forwarding set containing destinations that h_j should forward the PKT to, and $\mathcal{D}_\mathcal{N}$ as the broadcast set containing destinations that all neighbors should forward the PKT to (because no reliable route is known).

To take advantage of the wireless broadcast channel, routing decisions (UNICAST and BROADCAST) are accumulated and transmitted in a single MAC layer broadcast. The accumulated routing decisions are organized in a *forwarder map* $\mathcal{F} = \langle \langle h_1, \mathcal{D}_1 \rangle, \ldots, \langle h_m, \mathcal{D}_m \rangle, \mathcal{D}_\mathcal{N} \rangle$ with m being the number of neighbors we are transmitting to. A receiving neighbor h_j then filters for its respective $\mathcal{D}_j$ and, in turn, applies Algorithm 1 to PKT $(\mathcal{D} := \mathcal{D}_j \cup \mathcal{D}_\mathcal{N})$.

B. Efficient Per-Destination Subflows

In order to preserve the security of Castor, our explicit multicast protocol needs to maintain independent flow state for each destination. A naïve approach would be to include a full Castor header (flow ID, flow authenticator, PKT ID, and PKT authenticator) for each destination. This is *highly* inefficient and impractical even for small groups because the header size quickly grows to multiple kilobytes.

We propose a much more efficient solution incurring a bandwidth penalty of only one PKT identifier per destination. To this end, we introduce additional layers in the Merkle tree, where the exact number depends on the group size n. For PKT k of a certain flow, the source draws a value c_k at random. The source node leverages the shared keys K_{sd_i} to calculate per-destination ACK authenticators $a_{k,i}$. From $a_{k,i}$, the source derives per-destination PKT identifers $b_{k,i}$. The latter ones are in turn used to generate the leafs of the Merkle hash tree. The full generation is shown in Fig. 2.

When sending a PKT, the source includes c_k (can be seen as the equivalent e_k in Castor), as well as a list of all PKT identifiers B_k in the header:

$$B_k = \langle b_{k,1}, \ldots, b_{k,i}, \ldots, b_{k,n} \rangle \tag{1}$$

Upon PKT reception, destination d_i calculates $a_{k,i}$ from c_k and returns ACK $= \langle a_{k,i} \rangle$. Intuitively, the challenge at the destination is to encrypt a_k with the source–destination secret K_{sd_i}. This ensures that *only* the destination can acknowledge the reception of PKT with b_k. Note that even another group member cannot forge the ACK for d_i as it would require knowledge of K_{sd_i}. This assures immunity against insider adversaries.

Due to the nature of Xcast, some header fields of Xcastor, in particular $\mathcal{F}$ and B_k, change during transit. This means that these fields cannot be end-to-end integrity protected. However, they do not need this protection, as explained below:

Reordering Attack. Let an adversary swap any two destinations d_i and d_j in $\mathcal{F}$. This would go unnoticed by intermediate nodes. Upon reception, destination d_i returns the correct ACK with $a_{k,i} = \text{ENC}_{K_{sd_i}}(c_k)$ which is returned on the reverse path. The first correct node upstream of d_i receiving this ACK would realize that $\text{H}(a_{k,i})$ does not match the previously forwarded $b_{k,j}$. The ACK is consequently discarded, resulting in a reliability estimator penalty for the entire path including the adversary after T_{ACK}. Similar reasoning can be applied to show that other tampering with $\mathcal{D}$ will not break the protocol.

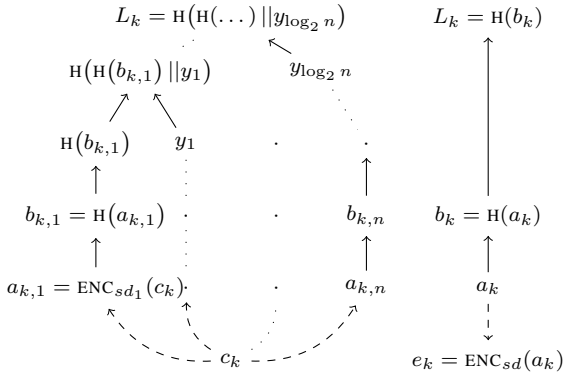

Fig. 2: Merkle tree leaf generation (L_k) in Xcastor (*left*) and Castor (*right*).

C. Packet Format

The packet format of our final Xcastor looks as follows:

$$\text{PKT} = \Big\langle s, \overbrace{\langle h_1, \mathcal{D}_1\rangle, \ldots, \langle h_j, \mathcal{D}_j\rangle, \ldots, \langle h_m, \mathcal{D}_m\rangle, \mathcal{D}_{\mathcal{N}}}^{\text{forwarder map } \mathcal{F}}, H, \underbrace{b_{k,1}, \ldots, b_{k,i}, \ldots, b_{k,n}}_{\text{per-destination PKT ids } B_k}, f_k, c_k, \mathcal{P}\Big\rangle, \quad (2)$$

$$\text{ACK} = \langle a_{k,i}\rangle, \quad (3)$$

where

$\bigcup_{j=1}^{m} \mathcal{D}_j \cup \mathcal{D}_{\mathcal{N}} = \mathcal{D}$ is the set of destinations,
H, f_k are the flow identifier and authenticator, respectively, as in Castor,
c_k are the seeds for the Merkle tree,
$a_{k,i} = \text{ENC}_{K_{sd_i}}(c_k)$ are the ACK authenticators, and
$b_{k,i} = \text{H}(a_{k,i})$ are the per-destination PKT identifiers.

D. Packet Processing

We next describe Xcastor's packet processing algorithm and highlight differences and similarities to the Castor protocol.

Forwarder map filtering. Upon PKT reception, a node x first checks whether it is included in the forwarder map. If not (i. e., if $\mathcal{D}' := \mathcal{D}_x \cup \mathcal{D}_{\mathcal{N}} \stackrel{!}{=} \emptyset$), the PKT is discarded. Otherwise, x removes all $\langle h_j, \mathcal{D}_j\rangle$ for $j \neq x$ and the corresponding $b_{k,i}$ values. Then continues with a **duplicate check**: the previously seen $b_{k,i}$ are removed along with d_i, and if a corresponding ACK for $b_{k,i}$ was already received, a copy of the cached ACK is returned to the PKT sender. Note that when removing $b_{k,i}$ from the PKT, some intermediate tree elements y (Fig. 2) must remain in B_k to allow proper verification of the Merkle tree (below).

Flow verification is essentially the same as in Castor: x needs to calculate L_k from B_k and then verify that L_k is in fact a leaf of the Merkle hash tree with root H using the sibling nodes in f_k. As an example, for $k = 1$, the check would look as follows (the concatenation order for other k depends on the position of b_k in the Merkle hash tree):

$$\text{H}\big(\ldots \text{H}(\text{H}(\text{H}(b_1)\,||x_1)\,||x_2)\,||\ldots x_{\log_2 w}\big) \stackrel{!}{=} H. \quad (4)$$

If the check fails, the PKT is discarded.

PKT verification. When a PKT arrives at destination d_i, it calculates $a_{k,i} = \text{ENC}_{K_{sd_i}}(c_k)$ and verifies that

$$\text{H}(a_{k,i}) \stackrel{!}{=} b_{k,i}. \quad (5)$$

If successful, d_i returns ACK $= \langle a_{k,i}\rangle$ to the PKT sender. Then, d_i removes itself and $b_{k,i}$ from PKT. If $\mathcal{D}'' := \mathcal{D}' \backslash \{d_i\}$ contains other destinations, d_i has to forward the PKT.

Castor nodes maintain routing state as a running average of the delivery reliability per flow and per neighbor (*reliability estimator*). In Xcastor, we introduce *subflows* $H_i = \langle H, d_i\rangle$, and keep reliability estimators per flow, per neighbor, *and* per destination, to allow independent routing decisions for each destination. The **PKT forwarding** process is as follows: (1) look up the next hop for every destination (subflow H_i) independently, (2) generate the forwarder map, and (3) start individual timers T_{ACK} for each forwarded $b_{k,i}$. If $b_{k,i}$ was forwarded over neighbor h_j and h_j does not return a valid ACK until T_{ACK} times out, then the appropriate reliability estimator is decreased.

ACK verification is similar to that of Castor. The only difference is that the destination calculates $b_{k,i} = \text{H}(a_{k,i})$ instead of $b_k = \text{H}(a_k)$ and checks whether it as previously forwarded $b_{k,i}$. If yes *and* T_{ACK} has not yet expired, the reliability estimator for the sending node h_j is increased.

VI. Implementation

Our implementation builds upon the Click modular router framework [21]. In the following, we discuss some implementation details that strongly influence the performance of Xcastor.

Cryptographic primitives. The choice of cryptographic primitives is relevant for the bandwidth efficiency of our protocol. We use SHA-1 and AES-128-CTR. This keeps hashes and ciphertext values as low as 20 bytes. For a flow size of 256, the total security overhead per PKT is 200 (constant), plus 20 bytes per destination (individual PKT identifier).

ACK timeout. The authors of [5] employ a fixed ACK timeout for updating Castor's reliability estimators. This impedes optimal operation for two reasons: firstly, reliability estimators should be updated as fast as possible, i. e., ACKs on low-delay paths should expire earlier. Secondly, round-trip times increase in highly loaded networks due to contention on the MAC layer. Castor will discard late ACKs leading to subsequent broadcasts for route discovery. We implement an adaptive timeout similar to TCP [22], exploiting the "free" ACKs of Castor and Xcastor.

Semi-reliable link-layer multicast. By default, Xcastor uses IEEE 802.11 broadcasts to address all nodes in the forwarder map. To overcome the unreliability of 802.11 broadcasts (no retransmissions [23]), we implement a semi-reliable multicast scheme: all wireless interfaces are set into promiscuous mode. Then, the forwarding node unicasts to one of its neighbors $\bar{h}$ from the forwarder map. Upon loss, the sender will retransmit, giving $\bar{h}$ *and* all other neighbors a second chance to overhear the packet. For maximal reliability, $\bar{h}$ is chosen to be the node which has the largest forwarder set assigned. This scheme is *semi-reliable*, because the link-level reliability is achieved in a best-effort manner: a lost transmission to $h_j \neq \bar{h}$ is not recognized. The simulation results show that this simple, noninvasive scheme is sufficient to provide a level of reliability similar to Castor operating as a multicast-via-unicast protocol.

VII. Evaluation

We conduct simulation-based experiments using ns-3.22 [24] as our simulator of choice due to its Click integration. We benchmark Xcastor against the baseline Castor unicast protocol with the same security properties operating as a multicast-via-unicast protocol (i. e., it unicasts a PKT to each group member), and a simple flooding protocol which employs

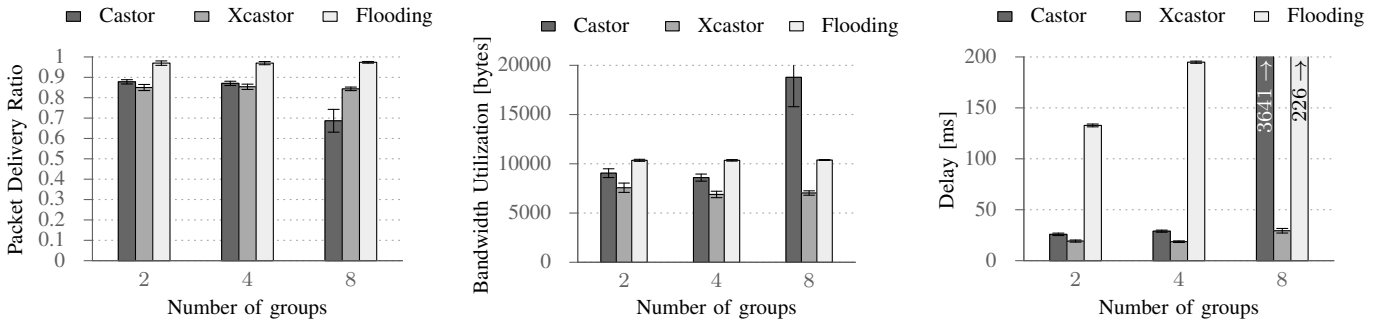

Fig. 3: Protocol performance with different **numbers of groups** actively sending packets.

TABLE I: Simulation settings

Network	Dimensions $w \times h$	$3000 \times 3000\,\mathrm{m}^2$
	Number of nodes N	100
Traffic	Group size	5
	Payload size and rate	512 bytes per 0.5 s
Mobility	Steady state random waypoint	$v \in [1, 10]$ m/s, $t_{\text{pause}} = 0$ s
Castor	Broadcast adjust γ	8.0 (as in [5])
	Update delta δ	0.8 (as in [5])
	Adaptive ACK timeout T_{ACK}	$[0.1, 60]$ s, init. 1 s
	Flow size 2^l	256 ($\Rightarrow$ lifetime 128 s)
802.11a MAC	Beaconing	Interval 0.25 s, timeout 1 s
	Max. retransmission count	7
	RTS/CTS threshold	256 bytes
PHY	Propagation loss model	Friis (free-space path loss)
	$\Rightarrow$ Transmission range r	560.3 m

no security mechanisms but is agnostic to wormhole/blackhole attacks since it is fully stateless. Flooding is known to have excellent reliability even under extreme conditions at the cost of very high bandwidth utilization. We omit benchmarks against other multicast protocols because non of the existing ones are comparable to Xcastor w. r. t. their security properties. Our metrics of interest are (a) raw packet delivery ratio (PDR), i. e., the fraction of successfully received PKTs without any network or upper layer retransmission scheme (*reliability*), (b) bandwidth utilization (BU) per PKT and per destination[1] (*efficiency*), and (c) end-to-end delivery delay (*speed*).

Detailed simulation settings for ns-3 are provided in Table I. The plots show averages over 20 experiment runs, seeded with numbers from the interval $[1, 20]$. The error bars indicate 95 % confidence intervals. We first summarize our results and then discuss them in detail:

- *Scalability* (VII-A): Castor collapses under high load when introducing more groups into the network. Xcastor is only marginally affected.
- *Security*: (VII-B): Xcastor and Castor maintain close to identical reliability under the strong greyhole attack, while Xcastor is more efficient. We identify flow restarts as a security problem for both protocols.

[1]The bandwidth utilization includes all frames that are transmitted through a node's wireless interface during the run of the simulation, i. e., it accounts for PKTs, ACKs, beacons, MAC control frames, and retransmissions.

A. Scalability: Number of Groups

Xcastor is supposed to scale well with the *number of groups* in the network. We quantify this by letting up to 8 sources simultaneously transmit to 5 destinations each (Fig. 3), which means that up to 48 nodes are involved in end-to-end data transmissions. Xcastor seems to be agnostic to the increased load: PDR and BU are not affected, only the delivery delay slightly increases. Castor collapses under the load due to congestion at a group size of 8. The result is a vicious circle: less ACKs are coming through, which causes the protocol to broadcast more often ($> 80\,\%$ of routing decisions are broadcasts). Because of the high broadcast rate, each PKT is effectively flooded 5 (!) times, because Castor establishes individual flows for each of the 5 destinations. Interestingly, PDR of flooding remains largely unaffected by the higher load: the high packet redundancy in the network compensates for packet collisions. However, this comes at the cost of linearly increasing delivery delays, caused by the strong contention on the MAC layer.

B. Security: Attack Resilience

So far, we have considered the benign case, i. e., without adversaries present in the network. In this experiment, we quantify the attack resistance by introducing different percentages of malicious nodes into the network (Fig. 4). Malicious nodes conduct a greyhole attack as described in [5], i. e., they forward all broadcast PKTs to attract traffic and then drop subsequent unicast PKTs. ACKs are always forwarded. According to [5], wormhole, blackhole, sybil, rushing, and jamming attacks are less severe than the greyhole attack. Thus, we focus on the strongest attack in the evaluation. In this setting, sources and destinations are considered to be benign. Malicious nodes are chosen uniformly at random from the remaining ones.

For the flooding protocol, a malicious node simply drops all traffic (blackhole). The effect is the same as if up to $40\,\%$ of the nodes were removed from the network. By not forwarding any packets, the malicious nodes basically exclude themselves from the network. Ergo, BU drops linearly. PDR decreases slightly because the effective node density declines, but remains at a high level. The impact of attackers on Xcastor is not severe: with $20\,\%$ of all nodes as attackers, PDR is

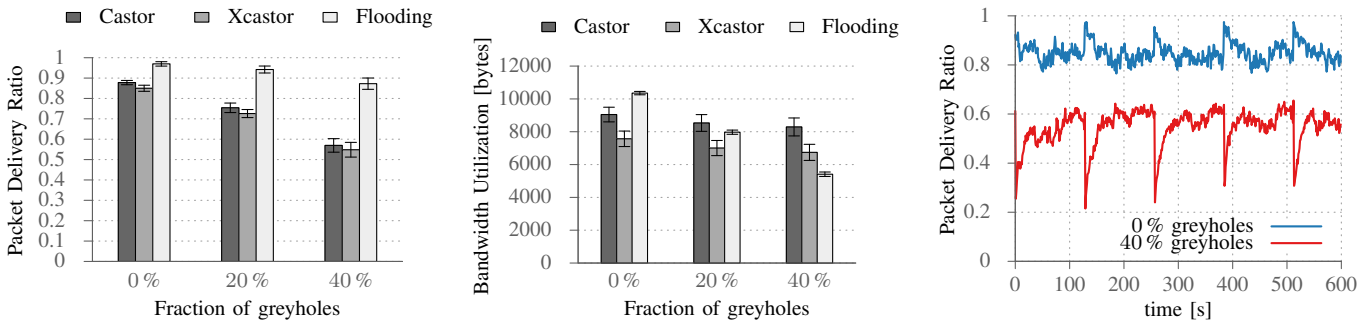

Fig. 4: Protocol performance under the **greyhole attack**. The right plot shows Xcastor's behavior over time. The data points are one-second averages.

reduced by about 15 %. The effect becomes more pronounced if we double the amount of malicious nodes in the network.

Flow restarts in adverse environments. The authors of Castor claim that flow restarts (establishing a new flow when the old PKT identifiers are exhausted) are practically a non-issue if the flow size is sufficiently large [5]. This is true when considering bandwidth utilization: each flow initialization requires flooding of a few PKTs. After a few successfully received ACKs, the protocol starts unicasting because reliable routes have been found. We show the behavior of PDR over time in the benign vs. the adversarial case in Fig. 4 (right): the periodic peaks indicate the short flooding periods for flow establishment. After a few successfully received ACKs, the nodes start unicasting. In the benign case, PDR decreases slightly and stabilizes quickly. In the adverse case, the initial unicasts lead to an excessive amount of PKT drops (PDR < 0.3), followed by a slow convergence towards a relatively stable value around 0.6. By "forgetting" the old state, Xcastor suffers a reliability penalty because adversary-free routes have to be rediscovered after each flow restart. Increasing the flow size can mitigate this problem, however, the header size poses a limit as the flow authenticator f_k is dependent on the flow size.

VIII. Conclusion

We have explored the feasibility of secure small group multicast in mobile ad hoc networks. We have designed and implemented an explicit multicast protocol based on Castor which we call Xcastor. Based on ns-3 simulations, we have shown that Xcastor performs equally well as Castor from a security perspective; and Xcastor can support more groups concurrently than Castor and flooding, which both experience much higher delivery delays. The experiments were conducted in a mobile scenario.

Finally, we would like to point out some directions for future work: (a) we have discovered that flow restarts have a significant impact on Xcastor's and on Castor's reliability. We are currently working on a solution that completely avoids flow restarts without relying on public-key cryptography. (b) An Xcastor node might transmit the same PKT multiple times if it was selected as a forwarder by different neighbors at different points in time. One solution could be to not immediately forward each PKT but wait for other copies to arrive and then serve all destinations with a single transmission.

References

[1] BBC News. (2014, Jun.) Iraqis use Firechat messaging app to overcome net block. [Online] http://www.bbc.com/news/technology-27994309

[2] H. Goldstein. (2010, Jan.) Engineers race to restore communications after Haiti quake. [Online] http://spectrum.ieee.org/tech-talk/telecom/internet/engineers-race-to-restore-communications-after-haiti-quake

[3] P. Gupta and P. R. Kumar, "The capacity of wireless networks," *IEEE Trans. on Inform. Theory*, vol. 46, 2000.

[4] J. Andrews *et al.*, "Rethinking information theory for mobile ad hoc networks," *IEEE Commun. Mag.*, vol. 46, 2008.

[5] W. Galuba *et al.*, "Castor: scalable secure routing for ad hoc networks," in *Proc. IEEE INFOCOM*, 2010.

[6] E. M. Royer and C. E. Perkins, "Multicast operation of the ad-hoc on-demand distance vector routing protocol," in *Proc. ACM MobiCom*, 1999.

[7] J. J. Garcia-Luna-Aceves and E. L. Madruga, "The core-assisted mesh protocol," *IEEE J. on Select. Areas in Commun.*, vol. 17, 1999.

[8] S.-J. Lee, W. Su, and M. Gerla, "On-demand multicast routing protocol in multihop wireless mobile networks," *Mobile Networks and Applications*, vol. 7, 2002.

[9] J. Luo, P. T. Eugster, and J.-P. Hubaux, "Route driven gossip: probabilistic reliable multicast in ad hoc networks," in *Proc. IEEE INFOCOM*, 2003.

[10] R. Boivie *et al.*, "Xcast concepts and options," IETF, RFC 5058, 2007.

[11] L. Ji and M. S. Corson, "Differential destination multicast—a MANET multicast routing protocol for small groups," in *Proc. IEEE INFOCOM*, 2001.

[12] C. Gui and P. Mohapatra, "Scalable multicasting in mobile ad hoc networks," in *Proc. IEEE INFOCOM*, 2004.

[13] H. Gossain *et al.*, "A scalable explicit multicast protocol for MANETs," *IEEE J. on Commun. and Networks*, vol. 7, 2005.

[14] S. Roy *et al.*, "Securing MAODV: attacks and countermeasures," in *Proc. IEEE SECON*, 2005.

[15] R. Curtmola and C. Nita-Rotaru, "BSMR: byzantine-resilient secure multicast routing in multihop wireless networks," *IEEE Trans. on Mobile Computing*, vol. 8, 2009.

[16] M. G. Zapata and N. Asokan, "Securing ad hoc routing protocols," in *Proc. ACM Workshop on Wireless Security (WiSe)*, 2002.

[17] P. Papadimitratos and Z. J. Haas, "Secure routing for mobile ad hoc networks," in *Proc. SCS CNDS*, 2002.

[18] K. Sanzgiri *et al.*, "A secure routing protocol for ad hoc networks," in *Proc. IEEE Int. Conf. on Network Protocols (ICNP)*, 2002.

[19] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Ariadne: a secure on-demand routing protocol for ad hoc networks," *Wireless Networks*, vol. 11, 2005.

[20] D. Dolev and A. C. Yao, "On the security of public key protocols," *IEEE Trans. on Inform. Theory*, vol. 29, 1983.

[21] E. Kohler *et al.*, "The Click modular router," *ACM Trans. on Comput. Syst.*, vol. 18, 2000.

[22] V. Paxson *et al.*, "Computing TCP's retransmission timer," IETF, RFC 6298, 2011.

[23] IEEE Computer Society, "Part 11: wireless LAN medium access control (MAC) and physical layer (PHY) specifications," IEEE Std 802.11-2012.

[24] NS-3 network simulator. [Online] https://www.nsnam.org/