Attack-resistant Spanning Tree Construction in Route-Restricted Overlay Networks

Martin Byrenheid
TU Dresden
martin.byrenheid@tu-dresden.de

Stefanie Roos
Delft University of Technology
s.roos@tudelft.nl

Thorsten Strufe
TU Dresden
thorsten.strufe@tu-dresden.de

***Abstract*—Nodes in route-restricted overlays have an immutable set of neighbors, explicitly specified by their users. Popular examples include payment networks such as the Lightning network as well as social overlays such as the Dark Freenet. Routing algorithms are central to such overlays as they enable communication between nodes that are not directly connected. Recent results show that algorithms based on spanning trees are the most promising provably efficient choice. However, all suggested solutions fail to address how distributed spanning tree algorithms can deal with active denial of service attacks by malicious nodes.**

In this work, we design a novel self-stabilizing spanning tree construction algorithm that utilizes cryptographic signatures and prove that it reduces the set of nodes affected by active attacks. Our simulations substantiate this theoretical result with concrete values based on real-world data sets. In particular, our results indicate that our algorithm reduces the number of affected nodes by up to 74% compared to state-of-the-art attack-resistant spanning tree constructions.

I. Introduction

Payment or state channel networks like Lightning [14] are the most promising approach to scaling blockchains, i.e., enabling blockchain-based payment systems to process tens of thousands of transactions per second with nearly instant confirmation. Participants in such payment networks establish channels for trading assets such as digital coins. As establishing channels requires use of the blockchain, which is both time- and cost-intensive, only nodes that frequently trade with each other establish payment channels [7]. All other payments pass from a sender to the receiver via multi-hop paths of channels. It is essential to find these paths in an effective, efficient, and privacy-preserving manner [17].

Similarly, social overlays require finding paths from a peer to another in a network consisting only of connections between trusted pairs of nodes to realize scalable and privacy-preserving distributed services [4], [16].

Both payment channel networks and social overlays hence share the need for a routing algorithm. A number of promising algorithms for both networks rely on Breadth-First-Search (BFS) spanning trees [12], [16], [17], as these permit finding shortest paths and achieve the most efficient communication. The underlying spanning tree construction algorithm determines the effectiveness, efficiency, and attack resilience of the routing. Resistance to attacks by malicious parties who aim to prevent the tree construction from converging towards a correct spanning tree is particularly important. Preventing the construction of a correct spanning tree results in routing failures and hence constitutes a denial-of-service attack that undermines communication. Such attacks are realistic for both payment channel networks and social overlays. For payment channel networks, adversarial parties may undermine the routing of payments to sabotage competing operators. Social overlays such as Freenet aim to protect communication from censorship [4]. They clearly require attack resistance against participants aiming to execute censorship in the form of a denial-of-service attack.

In the context of route-restricted overlays with potentially malicious participants, spanning tree algorithms have to fulfill three requirements: (1) enable efficient communication by providing short paths between honest nodes in the spanning tree, (2) efficiently adapt to changes of the network structure, and (3) maintain high availability in the presence of malicious nodes that deliberately deviate from the construction protocol in order keep the network from converging. Yet, the existing work on spanning tree-based routing only evaluates the first two aspects jointly, leaving protection against malicious behavior out of scope despite the likely existence of malicious parties in both payment channel networks and social overlays.

In this work, we focus on achieving all three requirements jointly, giving rise to two key contributions:

- We present a self-stabilizing algorithm for the computation of a BFS spanning tree that uses cryptographic signatures to check the integrity of statements about the distance to the root node. We prove that the fraction of nodes reaching a stable, non-compromised state is higher than in state-of-the-art protocols.
- We present results from an extensive simulation study based on real-world data sets. The results demonstrate that the construction of BFS spanning trees without cryptographic measures is highly vulnerable to attacks, even if the adversary establishes just a handful of connections to honest nodes. Furthermore, we show that our algorithm substantially raises the necessary number of such attack connections to mislead a comparable number of nodes.

II. Related Work

We review the existing work for routing in route-restricted overlays to show that the design of attack-resistant spanning trees is indeed the key problem to solve. Afterwards, we consider the existing work on attack-resistant spanning tree

2575-8462/19/$31.00 ©2019 IEEE
DOI 10.1109/SRDS47363.2019.00035

constructions, which we then improve upon in the following sections.

A. Routing in route-restricted overlays

We define an *overlay network*, or just overlay, as a network between multiple logically connected nodes that communicate via a public infrastructure such as the Internet. In *route-restricted overlays*, the logical connections between nodes are explicitly managed by their respective users and hard or even impossible to adapt to create a topology that benefits routing. Apart from finding existing paths between nodes, routing algorithms have to be efficient and scalable with regard to delays for the delivery of messages, bandwidth and memory consumption to provide adequate service for large-scale peer-to-peer networks such as payment channel networks and social overlays. Recent work [12], [16], [17] underlines that only routing algorithms based on rooted spanning trees provide the necessary efficiency. Other approaches either use expensive flooding for path discovery [11] or setup virtual tunnels [13], [15], [22], which, in face of network dynamics, require costly maintenance [18]. Alternatively, some payment channel networks of smaller size use source routing [14], [19], which requires that each node maintains a snapshot of the entire network. Source routing hence does not scale, as any change to the network has to be broadcast.

In the context of social overlays, Hoefer et al. [9] suggested using greedy embeddings based on rooted spanning trees to enable efficient routing between nodes. The approach has later been extended to preserve the privacy of users and offer higher attack resistance [16]. However, their adversarial model only considers the routing and not the construction of the underlying spanning tree, which is an orthogonal approach to the one taken in this paper.

For payment channel networks, Malavolta et al. [12] adapted Landmark Routing [20], where a path between sender and receiver is determined through an intermediate node via the construction of a breadth-first-search tree rooted at the latter. Roos et al. later on adapted the greedy embeddings to payment channel networks [17]. Both works aim to achieve efficiency and privacy and do not consider security.

It thus remains an open question, if and how such spanning trees can be constructed in route-restricted overlays with malicious participants.

B. Attack-resistant spanning tree construction

In the context of self-stabilization, Dubois, Masuzawa, and Tixeuil proposed a BFS spanning tree algorithm and proved that this algorithm guarantees that all nodes, except those that are strictly closer to the adversary than to the root node, will eventually converge to a correct state [6]. While the algorithm by Dubois et al. offers provable attack resistance, it considers a computationally unbounded attacker. Protecting against such a strong adversary disregards mechanisms such as digital signatures that can help to further decrease the number of affected nodes.

In the context of distance vector routing, which implicitly relies on BFS trees, Zapata and Asokan [24] proposed a protocol that utilizes hash chains to keep malicious nodes from lying about their distance from the root node. Furthermore, their protocol employs cryptographic signatures to prevent attacks on the mechanism for the detection of routing loops. Subsequently, Hu et al. [10] proposed a protocol that uses hash chains both against attacks on the reported distance as well as against attacks on loop-detection, thus reducing computational overhead compared to digital signatures. In contrast to the work of Dubois et al., both approaches assume a computationally bounded attacker. However, they do not provide a formal proof of their security guarantees.

In summary, there exists no provably secure BFS tree construction algorithm under the assumption of a computationally bounded attacker. We expect that such an algorithm can provide protection to a larger set of nodes than the existing information theoretically secure algorithms.

III. Model and Notation

We now formalize route-restricted overlays as well as the problem of computing a breadth-first-search tree in the context of self-stabilization.

A. System model

We model a route-restricted overlay $S = (V, E)$ as a finite set V of n nodes and a set of bidirectional communication links $E \subset V \times V$. For each node u, the set $N(u) = \{v \mid \{u, v\} \in E\}$ denotes the *neighbors of* u.

We build upon the shared memory model where each pair of nodes $\{u, v\} \in E$ can communicate via shared registers r_{uv} and r_{vu}, where u is only allowed to write into r_{uv} and read from r_{vu}. We thus call r_{uv} u's *output register* and r_{vu} its *input register*.

Please note that we use the shared memory model solely to simplify formal analysis, as it omits the modeling of message transmission. We consider this to be reasonable, as we focus on malicious node behavior and neither link failures nor delays. For the computation of a BFS tree, every node u holds the following elements:

- ID_u, a fixed, globally unique ID from a set $\mathcal{ID}$,
- $level_u$, a non-negative integer variable denoting u's current, assumed distance to the root,
- pID_u, a variable holding the ID of the node that is currently considered parent, in other words, the neighbor of u on the path to the root in the subgraph corresponding to the tree.

Furthermore, each communication register holds two values ID and $level$ such that each output register of a node u holds u's ID as well as its current $level$-value. Each input register r_{vu} of a node u accordingly holds u's current view of v's ID and $level$-value. In the following, we denote the set $N_{min}(u) = \{v \in N(u) | \forall n \in N(u) : level_v \leq level_n\}$ as *minimal neighbors* of u. Parent nodes are always minimal neighbors in BFS spanning trees.

We refer to the values currently held by the $level$- and pID-variable of a node u as well as the register contents, at one point in time, as the *state* of u. The state of u is said to be *legitimate* if it fulfills Def. 1.

Definition 1. (Legitimate state) *Let $S = (V, E)$ be a route-restricted overlay with a distinguished root node $l \in V$ with ID-value $ID_L \in \mathcal{ID}$. The state of a node u whose minimal neighbors have level l_{min} is called* legitimate *if it fulfills the following conditions:*

1) $level_u = 0$ *iff* $ID_u = ID_L$
2) $level_u = l_{min} + 1$ *if* $ID_u \neq ID_L$
3) $pID_u = ID_u$ *iff* $ID_u = ID_L$
4) $\exists v_{min} \in N_{min}(u) : pID_u = ID_{v_{min}}$ *if* $ID_u \neq ID_L$

B. Adversary model

In this work, we consider adversaries who aim to perform large-scale denial of service attacks. For payment networks, they might be competing payment network operators who want to attract more users by rendering other networks unusable. For social overlays, the adversary might aim to weaken the privacy [1] or degrade utility so that users move to communication services with weaker privacy protection.

Allowing multiple adversaries to act in concert strictly increases their power. We hence assume a single, collective adversary who controls a set B of *malicious* (or *adversarial*) nodes and is able to set up a bounded number of connections between these malicious and *honest* nodes H. The motivation for these bounds is the difficulty of large-scale social engineering that will only be successful for a subset of participants.

During an attack, each malicious node may report incorrect data to the adjacent honest nodes in order to keep them from reaching or remaining in a legitimate state. Thus, malicious nodes may set their output registers arbitrarily and report different ID- and $level$-values to different neighbors.

However, we assume that the adversary does not know all honest nodes and their internal connections a priori. Hence, he cannot choose which nodes will be malicious or which nodes will connect to malicious nodes. Given that social overlay and payment networks are large-scale and dynamic distributed systems with participants from a multitude of countries, we consider this assumption to be realistic.

For all practical purposes, the Dolev-Yao model, which assumes an adversary who is limited to polynomial-time attacks – and hence unable to break secure cryptographic primitives – has been accepted as realistic [5]. Hence, we aim for algorithms that protect against adversaries that are polynomially bounded.

C. Formalization of resilience and performance

We formalize the attack resistance of a spanning tree construction protocol via the concept of topology-aware (TA) strict stabilization [6]. To do so, we express the state of every node in the overlay at one point in time as a *configuration* γ.

Following the idea of self-stabilization, we consider that every node starts in an arbitrary state. Thus, nodes may change their state over time to reach a legitimate state. The sequence of configurations $\gamma_0, \gamma_1, \ldots$ is called a *computation* Γ. The transition from γ_t to γ_{t+1} is called a *step* and corresponds to at least one node processing the data in its input register and writing corresponding data into its output register.

Note that self-stabilizing algorithms never terminate but repeatedly update their state and communication registers. However, a node executing a step may not actually change the values of its variables or output registers (e.g., because its current state is legitimate).

a) Network dynamics: Route-restricted overlays are dynamic: nodes may join and leave the system, connections between nodes are established or torn down over time. An overlay $S = (V, E)$ changes into an overlay $S' = (V', E')$ with a potentially different network size as a consequence of such events. According to literature, we call such changes *churn events*. To account for the fact that computations are defined for a fixed system S, a churn event interrupts a computation on S and starts a new computation on S'.

At the beginning of the new computation, all nodes in $V \cap V'$ have the same state as at the end of the computation on S, reflecting the fact that they cannot detect the change until they read from their registers. The remaining nodes in V' may start in an arbitrary initial state. In route-restricted networks, the initial state includes information about the register of neighbors, which the new node will eventually write to.

b) Containment of attacks: TA strict stabilization for a set $S_B \subset H$ of honest nodes denotes that every honest node u except those in the set S_B eventually reaches and remains in a legitimate state. We call the set S_B the *containment area of* S, because S_B (also called *lost nodes*) represents the part of the network where the adversary can keep the state of nodes from converging, whereas all nodes outside of S_B (called *safe nodes*) will eventually reach and remain in a legitimate state.

We now formalize the concept of a node having only honest ancestors on its path to the root.

Definition 2. (Root-directed path) *Given a route-restricted overlay S and a configuration γ, the root-directed path P_u of a node u is a finite sequence $v_1, v_2, \ldots, v_{n+1}$ of nodes in a legitimate state such that $v_{n+1} = u$ and $pID_{v_{i+1}} = ID_{v_i}$ for all $1 \leq i \leq n$ and either $pID_{v_1} = ID_{v_1}$ (the legitimate root) or v_1 is a malicious node. We call u* ill-directed *if v_i is malicious for any $1 \leq i \leq n$ and* well-directed *otherwise.*

As long as a node is ill-directed, it is subject to changes in the $level$-value reported by the adversarial node on its root-directed path. Thus, it is not guaranteed to remain in a legitimate state. However, an ill-directed node is not inherently a lost node, because it might eventually become well-directed as the execution proceeds.

We express the situation that a node's state has converged and remains unaffected by attacks as follows:

Definition 3. (Stable state) *The state of a node u is said to be* stable *if it is legitimate and u never changes its $level_u$- and pID_u-variable as long as no churn event occurs. In*

*particular, actions performed by malicious nodes do not affect u. A configuration γ is called S_B-*stable *if the state of every node in $V \setminus S_B$ is stable.*

We define a S_B-topology-aware-strictly-stabilizing (S_B-TA-strictly-stabilizing) algorithm as follows:

Definition 4. (S_B-TA-strictly-stabilizing algorithm) *A distributed algorithm $\mathcal{A}$ is S_B-TA-strictly-stabilizing if and only if starting from an arbitrary configuration, every execution contains a S_B-stable configuration.*

c) Time complexity: To be able to reason about the time complexity that a distributed algorithm requires to reach a legitimate state, we use the concept of *asynchronous rounds*. The first *asynchronous round* of a computation Γ is the shortest prefix Γ' of Γ such that each node has read from and wrote to all of its registers at least once. The second asynchronous round then is the first asynchronous round of the computation following Γ' and so on. In other words, the length of an asynchronous round corresponds to the maximum amount of time needed for the slowest node (regarding computational speed) to process its inputs and write the corresponding outputs.

IV. Signature-based computation of BFS trees

The state-of-the-art algorithm for the construction of BFS trees proposed by Dubois et al. [6] ensures that all honest nodes whose distance from the closest malicious node is higher or equal than their distance from the root will eventually reach a stable state. As the set of nodes that do not reach a stable state is often quite large for this algorithm, we investigate algorithms that achieve a higher number of stable nodes. In contract to previous work, we assume our adversary to be computationally bounded.

In our design, each node u holds a public/private key pair p_u, s_u of an asymmetric cryptosystem. The public key p_u of each node u is stored in the ID-register and the secret s_u is stored in a new register called $secret_u$. The given leader ID ID_L then is the public key of the corresponding root node, implicitly choosing it as leader. Nodes do not require global knowledge of all other nodes' keys.

a) Assumptions: Four assumptions underlie our design:

- There is an honest root node whose key is known to all nodes (e.g., bank in a payment network [12]).
- The clocks of any pair of nodes differ at most by a globally known constant Δ_C.
- The time needed for one iteration of each node's main loop is bounded by a globally known constant Δ_E.
- The delay needed until a value written into an output register is available in the corresponding input register is bounded by a globally known constant Δ_D.

The first assumption is in accordance with the existing literature on tree-based routing in route-restricted overlays [12], [16], [17]. The remaining assumptions allow us to compute expiration times for the data contained in the input register of each node, thus keeping malicious nodes from reporting outdated values obtained in previous computations.

b) Level attestation: To keep malicious nodes from lying about their distance to the root, we add a $levelAtt$-variable to each node u, which holds a finite sequence $P = (p_1, t_1, sig_1), (p_2, t_2, sig_2), \ldots, (p_n, t_n, sig_n)$ of tuples called a *level attestation*. The elements p_i, t_i, and sig_i denote a public key, a timestamp, and a cryptographic signature, respectively. We say that such a sequence is *valid for node u at time t* if the following conditions are satisfied:

1) $p_1 = ID_L$,
2) $\forall i \in \{1, .., n\} : t - t_i \leq \Delta_C + (\Delta_D + \Delta_E)(n - i + 1)$,
3) $\forall i \in \{1, .., n-1\} : sig_i$ is a signature over $p_{i+1}||t_i$ that is valid for p_i,
4) sig_n holds a signature over $ID_u||t_n$ that is valid for p_n,

where $a||b$ denotes the concatenation of a and b.

Condition (1) ensures that the first tuple of the attestation has indeed been generated by the root node. Condition (2) ensures that adversarial nodes cannot use obsolete attestations (e.g., from an earlier computation) forever. Condition (3) and (4) ensure that the signatures are computed correctly.

c) Link signatures: Additional to the level attestation, each node assigns a randomly chosen *neighbor ID* nID_v to each neighbor v once in the beginning of the algorithm. During the computation, every honest node tells each neighbor its respective neighbor ID. Whenever a neighbor of a node u transmits a new level attestation, it also has to send a corresponding *neighbor signature* that includes its neighbor ID assigned by u. Given a valid level attestation P with the last element (p, t, sig) and a cryptographic hash function h, a neighbor signature s is *valid for node u and neighbor v* if s is a valid signature over $nID_v||h(P)$ for p. This addendum keeps malicious nodes from sending a shortened version of an attestation received by an honest neighbor.

d) Adaptive neighbor preference: To ensure stabilization in the case that a node has multiple neighbors that are minimal according to Def. 1, each node u assigns a unique number between 0 and $|N(u)| - 1$ to each neighbor and chooses the minimal neighbor with the lowest number as parent. The number of the current parent is kept in a variable $prnt$. As the preferred neighbor may be ill-directed, the algorithm of Dubois et al. [6] adaptively changes which neighbor will be preferred whenever a node changes its parent. We implemented this strategy as follows: We add an offset counter $i_{start} \in \{0, .., |N(u)| - 1\}$ such that u traverses its neighbors from i_{start} to $(|N(u)| - 1) + i_{start} \mod |N(u)|$. Whenever a node u changes its parent from the neighbor with number $prnt$ to a neighbor with a number $prnt'$ that, counting from i_{start} with wraparound, comes after $prnt$, then u will set i_{start} to $prnt'$, thus favoring $prnt'$ over $prnt$ in the future. To compare nodes' positions a and b with regard to i_{start}, we say that $a \prec_{i_{start}} b$ if either i) $i_{start} \leq a < b$, ii) $b < i_{start} \leq a$ or iii) $a < b < i_{start}$. Informally, $a \prec_{i_{start}} b$ indicates that b be will be reached later than a when counting from i_{start} modulo $|N(u)|$.

e) Spanning Tree Algorithm: Algorithm 1 displays the pseudocode for our spanning tree construction algorithm: Each output register of every node u holds 5 elements, namely

Algorithm 1: Attestation-based spanning tree on node u

```
1  while true do
2      foreach i in N(u) do
3          lr_iu := read(r_iu)
4      ts := getCurrentTime()
5      i_start := i_start mod |N(u)|
6      if ID = ID_L then
7          pID := ID
8          level := 0
9          levelAtt := nil
10     else
11         parentFound := false
12         N_valid := {i ∈ N(u) |
             isValidAtt(lr_iu.levelAtt, lr_iu.level + 1) ∧
             isValidLink(lr_iu.levelAtt, lr_iu.sig_adj)}
13         level := min{lr_iu.level | i ∈ N_valid} + 1
14         foreach i in 1..|N(u)| do
15             j := i + i_start mod |N(u)|
16             if not parentFound and N(j) ∈ N_valid and
                 level = lr_ju.level + 1 then
17                 if prnt ≺_{i_start} j then
18                     i_start := j
19                 prnt := j
20                 pID := lr_ju.ID
21                 levelAtt := lr_ju.levelAtt
22                 parentFound := true
23     foreach i in N(u) do
24         sig_lvl := sign(lr_iu.ID||ts)
25         exAtt := append(levelAtt, (ID, ts, sig_lvl))
26         sig_adj := sign(lr_iu.nID||h(exAtt))
27         write(r_ui) := (ID, level, exAtt, nID_i, sig_adj)
```

the ID- and $level$-value of u as well as the $levelAtt$- and nID-value together with the neighbor signature sig_{adj} for the corresponding neighbor. The algorithm leverages the following cryptographic functions: The $sign$-function uses the key stored in the $secret$-register to compute a signature sig. The function h is a cryptographic hash function.

Every node periodically reads the content of each input register, processes the content, and writes corresponding outputs to output registers. The leader node first ensures that its pID- and $level$-value are set correctly (Line 7–8). Subsequently, it generates a level attestation for each neighbor and writes its own ID and $level$-value together with the respective nID-value, level attestation, and neighbor signature into the corresponding output register (Line 24–27). Because the $levelAtt$-variable is set to nil, the $append$-operation in Line 25 just returns its second argument.

During the processing stage (Line 11–22), an honest non-leader node recomputes its current pID-, $prnt$-, $level$- and $levelAtt$-value. It first checks the validity of the received level attestations and neighbor signatures and computes the set of valid neighbors in Line 12. The $isValidAtt$-function checks whether a given level attestation is valid, as defined above. If the given level attestation is valid, $isValidAtt$ further checks whether the length of the attestation equals the given level value and returns false in case of a mismatch. Given this check succeeds, the $isValidLink$-function checks if a given sig_{adj}-value is valid for the corresponding neighbor. If a parent node with a valid level attestation has been chosen, the node first checks if its previous parent became either non-minimal or its attestation became invalid and if so, sets i_{start} to j. It is possible that $prnt$ might hold a value larger than $|N(u)| - 1$ (e.g. because its former parent had this number and left the overlay). $prnt$ will then be set to j that holds a value from the range $\{0, .., |N(u)| - 1\}$ (Line 15). Afterwards, it sets its $prnt$-, pID- and $levelAtt$-value accordingly. Finally, the node computes the corresponding level attestation for each neighbor and writes it into the respective output register (Line 24–27).

V. Analysis

We prove that, given an honest root node r, Algorithm 1 is S'_B-TA-strictly-stabilizing with

$$S'_B = \{u \in H \mid \exists b \in B : d^B_{min} + d_S(b, u) - 1 \leq d_S(r, u)\} \tag{1}$$

where $d^B_{min} = \min_{b \in B} d_S(r, b)$. The "-1" stems from the fact that a malicious node can copy the outputs of an honest neighbor into its output registers (hence pretending to be its own predecessor), thus avoiding the need to append an attestation tuple and hence increase its maximum level.

Furthermore, let $d^H_S(u, v)$ denote the length of the shortest path between u and v in S that does not contain a malicious node. If no such path exists, we set $d^H_S(u, v) = \infty$. If malicious nodes repeatedly change their outputs in order to destabilize honest nodes, we show that our algorithm guarantees that all nodes in the set

$$S'_L = \{u \in H \mid \exists b \in B : d^B_{min} + d_S(b, u) - 1 < d^H_S(r, u)\} \tag{2}$$

eventually reach a stable state. Informally, we show that $S'_L \subset S'_B$ is the containment area for an adversary that focuses on disrupting convergence by changing its behavior. However, for an arbitrary adversary aiming to maximize the fraction of ill-directed nodes, we achieve only a smaller containment area of S'_B.

Since the system starts in an arbitrary state, a malicious node may initially hold a level attestation that is valid but for which no corresponding path in the overlay exists. We hence say that a level attestation $(p_1, t_1, sig_1), \ldots, (p_n, t_n, sig_n)$ is *consistent for node* u if it is invalid or if there exists a path $v_1, \ldots, v_n$ in the system such that (1) p_i is the public key of v_i for all $1 \leq i \leq n$ and (2) u either is a neighbor of v_n or both u and v_n are neighbors of a malicious node b. Otherwise, we say that the attestation is *inconsistent*. A configuration is called consistent if the $levelAtt$-values as well as the in- and output-registers of all nodes only contain consistent level attestations.

In the following, we assume that at the beginning of a computation at time t, all timestamps of every inconsistent attestation are at most $t + \Delta_C$. We consider this to be reasonable since $t + \Delta_C$ is the highest value that a honest node (including the root) may use as timestamp and thus a malicious node cannot have a valid attestation with a higher timestamp from a previous computation. As a consequence, every inconsistent attestation of length n becomes invalid after

at most $\Delta_C + (\Delta_D + \Delta_E)n$ time units. So, every route-restricted overlay S with diameter $diam(S)$ reaches a consistent configuration after at most $\Delta_C + (\Delta_D + \Delta_E)diam(S)$ time units.

A. Proof of S'_B-TA-strict stabilization

We start the actual proof by establishing key properties of level attestation to later leverage in the proof. In a nutshell, malicious nodes can only influence keys that are used after the d^B_{min}-th element of a valid and consistent level attestation P but before the $|P| - d^B_{u,min}$-th element with $d^B_{u,min} = \min_{b \in B}\{d_S(u,b)\}$. Based on this result, we can then show that a node is well-directed if their $levelAtt$-value is of length less than $d^B_{min} + d^B_{u,min} - 1$. Convergence to a stable state for all nodes in S'_B follows from the fact that the system at some point reaches a state when these nodes have a valid and consistent $levelAtt$-value with minimal levels and hence will not change parents anymore.

Lemma 1. *Let $P = (p_1, t_1, sig_1), \ldots, (p_n, t_n, sig_n)$ be a level attestation. Consider a node u such that sig_n is a signature over $ID_u||t_n$. At time t, we have $t - t_i \leq \Delta_C + (\Delta_D + \Delta_E) \cdot (n - i + 1)$ for all $1 \leq i \leq n$ and the computation has started at least $\Delta_C + (\Delta_D + \Delta_E) \cdot n$ time units before, so that P is consistent for u. If P is valid, then the following two statements hold:*

1) *For $j \leq d^B_{min}$, p_j is the public key of an honest node v and $d_S(v,r) < j$.*
2) *For $j > n - d^B_{min,u} + 1$, p_j is the public key of an honest node v and $d_S(v,u) \leq n - j + 1$.*

Lemma 1 follows by induction on j and the full proof can be found in the extended version of the paper [3].

Lemma 2. *Let the computation have started a least $\Delta_C + (\Delta_D + \Delta_E) \cdot n$ time units before and $u \in V \setminus S'_B$ be a node with a valid $levelAtt$-value of length $n < d^B_{min} + d^B_{u,min} - 1$. Then u is well-directed.*

Proof. Because $\Delta_C + (\Delta_D + \Delta_E) \cdot n$ time units have passed, the $levelAtt$-value of u is also consistent. By Lemma 1, the first d^B_{min} public keys have to belong to honest nodes and the last $d^B_{u,min} - 1$ keys have to belong to honest nodes as well. Hence, if $n < d^B_{min} + d^B_{u,min} - 1$, all keys p_j have to belong to an honest node v_j for $1 \leq j \leq n$. Set $v_{n+1} = u$.

u can only be ill-directed if at least one v_j has their pID-value set to a key provided by a malicious node. First, consider the case that $j < d^B_{min}$. By Lemma 1, $d_S(v_j, r) < d^B_{min} - 1$, meaning that v_j only has honest neighbors. Honest nodes only write their own keys in the register of their neighbors, so that v_j can hence only set its pID-value to one of their keys. Now, consider $j > d^B_{min}$, i.e., $n - j + 1 < n - d^B_{min} + 1 \leq d^B_{u,min} - 1$. According to Lemma 1, $d_S(v_j, u) \leq n - j + 1 < d^B_{u,min} - 1$. Again, v_j has only honest neighbors and can hence only set its pID-value to one of their keys.

It remains to consider the case $j = d^B_{min}$. By the first part of the proof, v_j is the only node that can have malicious neighbors. Assume that v_j has set its pID to a malicious neighbor b. For u's $levelAtt$ to correspond to a valid attestation, v_{j-1} has to sign $p_j||t_{j-1}$ resulting in sig_{j-1}, append $(p_{j-1}, t_{j-1}, sig_{j-1})$ to the attestation, and write the attestation to the register corresponding to the neighbor that wrote p_j to the register. Because v_{j-1} has only honest neighbors, the respective neighbor has to be v_j, the only honest node that would claim p_j as its key. So, for u's $levelAtt$-value to include $(p_{j-1}, t_{j-1}, sig_{j-1})$, v_j must have read the register and disseminated $(p_{j-1}, t_{j-1}, sig_{j-1})$ as part of a level attestation. Consequently, v_j is aware that v_{j-1} offers a root-directed path of supposed length $j - 2 \leq d^B_{min} - 2$. For v_j to choose a different parent, b has to produce a valid attestation $\widetilde{P} = (\widetilde{p}_1, \widetilde{t}_1, \widetilde{sig}_1), \ldots, (\widetilde{p}_l, \widetilde{t}_l, \widetilde{sig}_l)$ with $l \leq j - 1$ and $\widetilde{sig}_l$ being a signature over $ID_{v_j}||\widetilde{t}_l$. Furthermore, b has to ensure that the $isValidLink$-function returns $true$. The neighbor-related signature has to be signed by the secret key $\widetilde{s}_l$ corresponding to $\widetilde{p}_l$. As b can not forge signatures, $\widetilde{P}$ has to be a (potentially shortened) attestation that b has read from one of its input registers. For such an attestation, $\widetilde{p}_l$ belongs to an honest node w at distance at most $l - 1$ from the root by Lemma 1. Due to $d^H_S(w,r) \leq l - 1 < d^B_{min} - 1$, w has no malicious neighbors. By Algorithm 1, w only writes signatures over $nID_w||h(L)$ for some L to registers of neighbors. Being honest, these neighbors do not disseminate the respective signatures. As a consequence, b can not obtain the required neighbor signature and hence v_j does not accept any attestation from b as its $levelAtt$-value.

In summary, none of the nodes v_j has its pID-value set to a key provided by a malicious node and hence u is indeed well-directed. □

Theorem 1. *Given any route-restricted overlay S with diameter $diam(S)$, a computation of Algorithm 1 starting from an arbitrary configuration reaches a consistent configuration after at most $\Delta_C + (\Delta_D + \Delta_E) \cdot diam(S)$ time units. Furthermore the computation will reach a S'_B-stable configuration γ^* within at most $diam(S) + 1$ additional asynchronous rounds. Thus, Algorithm 1 is S'_B-TA-strictly-stabilizing.*

Proof. Arrival at a consistent configuration follows from the assumption that the timestamps of every inconsistent level attestation do not exceed the starting time of the computation by more than Δ_C time units, as explained at the beginning of this section. To prove the subsequent convergence to a S'_B-stable configuration, we first show that after $l + 1$ rounds, all nodes $u \in V \setminus S'_B$ within distance l of the root are well-directed and have valid $levelAtt$-values of length l. The properties from Definition 1 follow. Last, we show that these nodes remain well-directed.

After the first round, the root has written its information to all registers. After the second round, the neighbors of the root have processed these registers. Hence, each such neighbor u will set its $levelAtt$-value to a valid attestation of length 1. If $u \in V \setminus S'_B$, the distance $d_S(u,b) \geq 2$ for any malicious node b and hence by Lemma 2, u is well-directed. So, the claim holds for $l = 1$.

Assume the claim holds for l, i.e., after $l+1$ rounds, all nodes $v \in V \setminus S'_B$ within distance l of the root are well-directed and have valid $levelAtt$-values of l. They know the IDs their neighbors have assigned to them as $l > 1$ indicates that they have read it from the register at least once. As a consequence, they can construct a valid attestation of length $l+1$ for each neighbor w as well as the necessary signature over the neighbor ID nID_w. They write this information to the register r_{vw}. After $l+1$ rounds, any node $u \in V \setminus S'_B$ at distance $l+1$ from the root has read the register corresponding to its neighbors at distance l to the root. As a consequence, u's $levelAtt$-value is of length $l+1$. As $u \in V \setminus S'_B$, Lemma 2 shows that u is well-directed. It follows by induction that within $diam(S)$ rounds, all nodes $u \in V \setminus S'_B$ are well-directed.

It remains to prove that the nodes in $V \setminus S'_B$ remain well-directed. To become ill-directed, a node has to change its pID-value. Let u be the first node to change its pID-value. According to Algorithm 1, u selects the parent from those neighbors that provide the shortest valid attestation and a valid neighbor signature. By assumption, u breaks ties consistently, meaning u only changes its parent if either i) u's previous parent does not provide any valid attestation of the shortest length or provides an invalid neighbor signature, or ii) a neighbor that is not the current parent writes an attestation of a shorter length than u's $levelAtt$-value to its register and the content of the register passes the two validity checks.

In order to conclude that neither i) or ii) are possible, consider the following: Let v be u's parent and note that $v \in V \setminus S'_B$ by the definition of S'_B as $d_S(v,r) = d_S(u,r) - 1$ and $d_S(v,b) \geq d_S(u,b) - 1$ for all malicious nodes b. It follows recursively that all nodes on a root-directed path of u are in $V \setminus S'_B$. Case i) would imply that a node on the root-directed path changed its parent, as honest nodes do not write invalid attestations or neighbor signatures to registers. However, such a parent change contradicts the definition of u as the first node in $V \setminus S'_B$ to change its parent. If case ii) holds, by Lemma 1, u has to be well-directed after its parent change. Hence, its new parent w is an honest node. By the above, w and all nodes on the new root-directed path are in $V \setminus S'_B$ and at least one of them has to have changed its parent for w to write an attestation of a different length. Again, such a change in parent is a contradiction to the definition of u. Consequently, nodes $u \in V \setminus S'_B$ do not change their pID-value for the rest of the computation and remain well-directed. □

B. Proof of stabilization for S'_L under attacks

Building upon Theorem 1, we now show that under an attacker that frequently changes the output values of its nodes, all nodes u with $d^B_{min} - d^H_S(b,u) - 1 = d^H_S(r,u)$ eventually reach a stable state as well. Our result requires the concept of a S_B-disturbance, a concept similar to Dubois et al. [6]'s S_B-disruption.

Definition 5. (S_B-disturbance) *Two consecutive configurations γ_0 and γ_1 are a S_B-*disturbance *if at least one node $u \in V \setminus S_B$ changes its $level_u$- or pID-variable.*

Table I
STRUCTURAL PROPERTIES OF GRAPHS USED FOR SIMULATION, WITH AVG. SHORTEST PATH LENGTH (CPL) AND CLUSTERING COEFFICIENT (CC).

Graph	# nodes	# edges	CPL	CC
Facebook	63,392	816,886	4.32	0.253
Ripple		99,787	3.82	0.154
Randomized Facebook		816,886	3.58	0.005
Erdös-Renyi		824,096	3.74	< 0.001

In contrast to a S_B-disruption, a S_B-disturbance does not assume that all nodes in $V \setminus S_B$ have a legitimate state.

Theorem 2. *Given any route-restricted overlay S with diameter $diam(S)$ and $deg_{sum} = \sum_{u \in S'_B \setminus S'_L} |N(u)|$, a computation of Algorithm 1 starting from an arbitrary configuration reaches a S'_B-stable configuration γ^+ within at most $\Delta_C + (\Delta_D + \Delta_E) \cdot diam(S)$ time units plus at most $diam(S)+1$ asynchronous rounds. After reaching the configuration γ^+, S will reach a S'_L-stable configuration within at most $(2deg_{sum} - |S'_B \setminus S'_L|)$ S'_L-disturbances.*

We only present a sketch of the proof here and refer the reader to the full version of the paper [3]. The S'_B-stability after at most $\Delta_C + (\Delta_D + \Delta_E) diam(S)$ time units and $diam(S)+1$ asynchronous rounds follows from Theorem1. Any subsequent S'_L-disturbance after reaching γ^+ can only affect nodes in $S'_B \setminus S'_L$. Let u be a node affected by a S'_L-disturbance. Algorithm 1 ensures that u changes its parent and the fact that $u \in S'_B \setminus S'_L$ means that u has a minimal neighbor v that offers a path consisting of all honest nodes to the root. After at most $|N(u)| - 1$ S'_L-disturbances affecting u, u starts increasing the variable i_{start} in Algorithm 1. From then on, u iterates over potential parents sequentially and increases i_{start} with each S'_L-disturbance, hence it will choose v as a parent after at most another $|N(u)|$ S'_L-disturbances that affect u. In total, at most $2|N(u)| - 1$ S'_L-disturbances affect u before it chooses v or another honest neighbor with a path of minimal length to the root. Afterwards, u does not change its parent as v does not change its level. The claim follows by summing over all nodes $u \in S'_B \setminus S'_L$.

VI. EVALUATION

Using OMNeT++ [21], we implemented a simulation to evaluate the impact of our attestation-based algorithm on the number of lost nodes compared to the non-cryptographic state-of-the-art. Furthermore, we investigated the impact of the network structure, the position of the root node, and the placement of edges between honest and malicious nodes.

A. Metrics, Data Sets, and System Parameters

Given a distributed system $S = (V, E)$ with a subset H of honest nodes and a S_B-TA stabilizing spanning tree construction algorithm, we measured the ratio of lost nodes (RLN) $|S_B|/|H|$. A low ratio of lost nodes indicates high attack resistance.

Route-restricted overlays include both social overlays and payment networks. We hence utilized a real-world graph for each of them and compare the results with synthetic

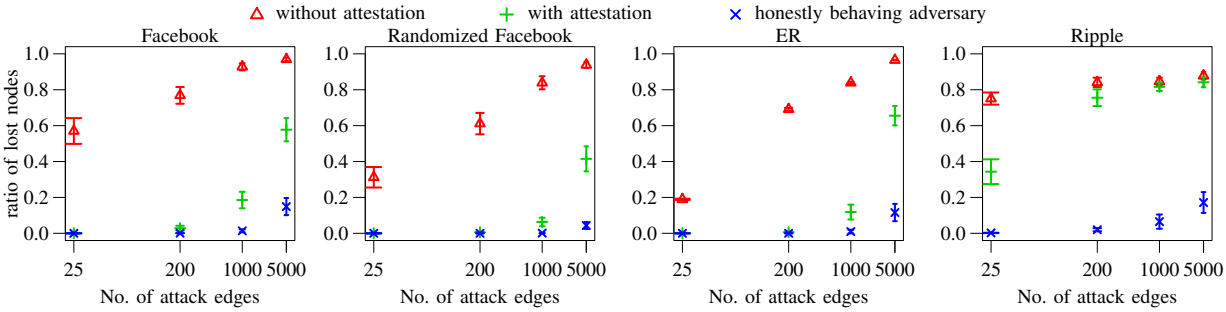

Figure 1. Observed mean ratio of lost nodes over 100 runs per configuration for 25, 200, 1000, and 5000 attack edges under the first adversarial behavior. The bars above and below each point represent 99% confidence intervals.

graphs for the purpose of characterizing the impact of various topological features. *Facebook* denotes a real-world graph of Facebook [23], as used in several prior studies [13], [16]. Ripple denotes a real-world graph from the Ripple payment network [17]. Ripple has a low number of edges and a heavily skewed degree distribution: 95% of all nodes have a degree less or equal than the average degree of approximately 3. Our synthetic data sets are i) a random synthetic network (denoted *randomized Facebook*) with the same degree distribution as the Facebook graph and ii) an Erdös and Renyi graph (*ER*) with approximately the same number of nodes and edges as *Facebook* but normal distributed degrees [8]. We compare *Facebook* with *randomized Facebook* to characterize the impact of clustering while the comparison of *randomized Facebook* and *ER* reveals the impact of the degree distribution.

We considered the number of malicious nodes and the time of their presence to be unbounded but limit the total number g of connections between honest nodes and malicious nodes. To model that all nodes are colluding, we represented them as a single node with g edges.

B. Set-up

We investigated the resistance of spanning tree algorithms to adversarial behavior given structural differences of the networks and a varying number g of attack edges. For all scenarios, we performed 100 runs to obtain statistically significant results.

We assumed the adversary knows all nodes but can only establish a connection to a subset with limited size. Following [2] we also assumed that users with many contacts are more likely to accept new requests and thus connect with a malicious node. We hence added a single adversary m to the graph and choose the g honest neighbors at random, with a probability proportional to their degree. Afterwards, a root node r was chosen uniformly at random from all honest nodes and the leader ID of each honest node was set accordingly.

We executed different spanning tree constructions for various adversarial behaviors. The two spanning tree algorithms are Algorithm 1, i.e., spanning tree construction with level attestation, and the state-of-the-art protocol by Dubois et al. [6]. The two adversarial behaviors are:

1) The attacker aims to prevent convergence by causing disturbances. By Theorem 2, the set of lost nodes corresponds to S'_L as defined in Eq. 2. Similarly, the set of lost nodes for the state-of-the-art protocol is $S_L = \{u \in H : d_S(u,m) < d_S(u,r)\}$ [6].
2) The attacker aims to maximize the number of ill-directed nodes. In this case, the adversary always pretends to be as close to the root as possible and does not perform any disturbances. In this case, the set of lost nodes is S'_B as defined in Eq. 1 according to Theorem 1. For the state-of-the-art protocol, the set of lost nodes is $S_B = \{u \in H : d_S(u,m) < d_S(u,r)\}$.

To investigate how strongly the cheating by one level (described in Sec. V) affects the number of lost nodes when Algorithm 1 is used, we furthermore simulated a modified adversary which does not cheat, effectively following Algorithm 1 correctly. As the results for the second adversarial behavior are very similar in terms of the advantage gained by Algorithm 1, we only present the results for the first adversarial behavior here. The remaining results are included in the extended version [3].

C. Impact of Level Attestation

Figure 1 show the obtained mean RLN with 99% confidence intervals for the four graphs and both algorithms under the first adversarial behavior. Especially for the Facebook graph, its randomized version, and the ER graph, Algorithm 1 considerably reduced the ratio of lost nodes compared to the state-of-the-art protocol. For the latter, an attack with 25 edges resulted in a mean RLN of 0.57, 0.31, 0.19, and 0.75 for the Facebook graph, the randomized Facebook graph, the ER graph and the Ripple graph, respectively. When applying Algorithm 1, the mean RLN at 25 attack edges dropped down to 0.0005, 0.00006, 0.00005, and 0.34 for the Facebook graph, the randomized Facebook graph, the ER graph, and the Ripple graph, respectively. Even for 1000 attack edges, the mean RLN for the Facebook graph, its randomized variant, and the ER graph significantly decreased from 0.93, 0.84, and 0.84 to 0.18, 0.06, and 0.12, respectively.

In the scenario with an adversary that does not cheat by a level, the mean RLN was considerably lower than in the scenario with Algorithm 1 alone, especially with 1000 and

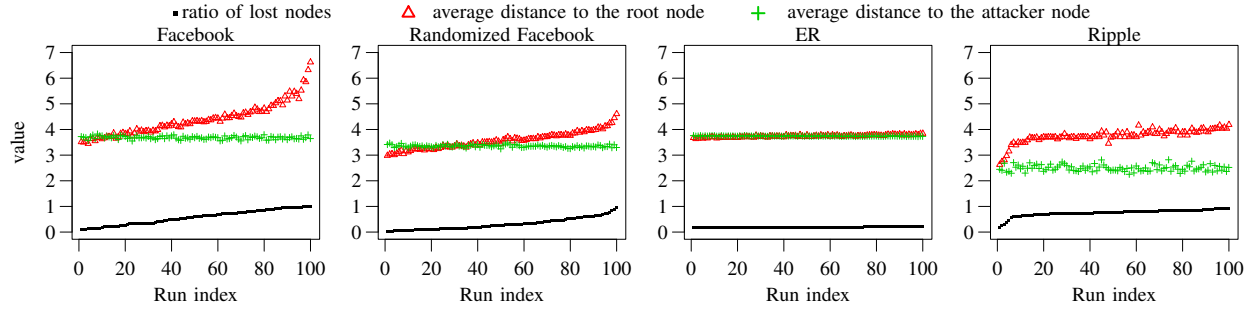

Figure 2. Obtained RLN values for the state-of-the-art protocol together with the average shortest path length to the root node and average shortest path length to the adversary node of the respective simulation run with 25 attack edges. The runs are ordered according to the RLN value in ascending order.

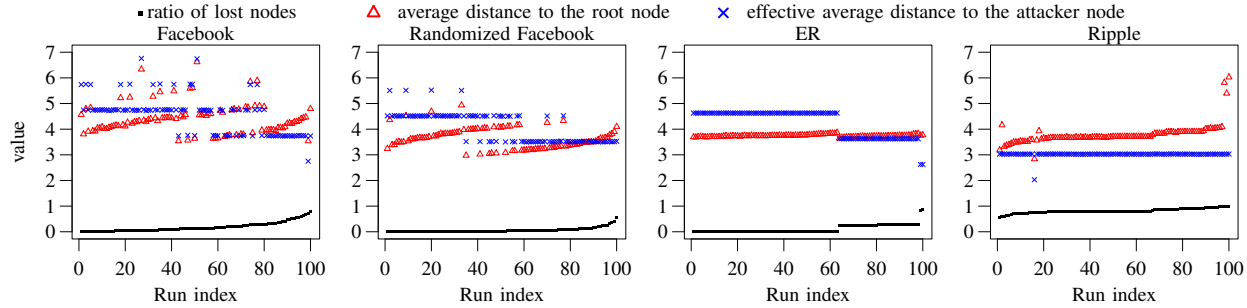

Figure 3. Results for the simulation runs of Algorithm 1 and 1000 attack edges. The *effective average distance* to m denotes the term $\overline{d}(m) + d(m, r) - 1$. The runs are ordered according to the RLN value in ascending order.

5000 attack edges. Referring to Table I we realize that all graphs used in our experiment have a very low average path length, and all nodes are in short distance from the root node. Increasing the adversary's reported $level$ value by 1 then represents a significant disadvantage for the attack. We conjecture that this causes many nodes to remain well-directed and investigate this relationship in more detail in the following.

For the Ripple graph, the improvement regarding mean RLN was considerably lower than for the other graphs. In the following, we describe the impact of distances between honest nodes, malicious nodes, and the root node on the RLN to explain this stark difference.

D. Impact of Network Structure

We start with a discussion of our results for the state-of-the-art algorithm and subsequently present results for Algorithm 1. Because the correlations between the different aspects were similar for both adversarial behaviors, we focus on our results for the first adversarial behavior.

a) State-of-the-Art Spanning Tree Construction: We considered the average hop distance over all honest nodes to the root node $\overline{d}(r)$ and to the attacker node $\overline{d}(m)$ for each simulation run. The lower $\overline{d}(r)$ is compared to $\overline{d}(m)$, the more honest nodes will have a lower hop distance to r than to m and thus be well-directed. Therefore, we expected a positive correlation between $\overline{d}(r) - \overline{d}(m)$ and the RLN.

Figure 2 shows the obtained RLN values in ascending order together with the corresponding value of $\overline{d}(r)$ and $\overline{d}(m)$ for 25 attack edges and both adversarial behaviors. Indeed, the difference $\overline{d}(r) - \overline{d}(m)$ generally correlated with the RLN. While $\overline{d}(m)$ only varied slightly between the different runs on each graph, there are notable differences in the behavior of $\overline{d}(r)$: It varied highly for the Facebook graphs and to some extent for the Ripple graph but barely for the ER graph.

The reason for the small variance in $\overline{d}(r)$ for ER is due to the uniform probability of two nodes being connected. As a consequence, it was very unlikely that the average distance of any node significantly differs from the other nodes. The degree of m, corresponding to the 25 attack edges, was close to the average degree of 26. However, the mean RLN was only 0.19, because there was a high number of nodes whose distance to the root node equalled that to the malicious node. These nodes chose the path to the root node when the malicious node continuously causes disturbances. In the setting where the attacker aims to maximize the number of ill-directed nodes, approximately half of all nodes became ill-directed.

For the Facebook graphs and Ripple, there is a higher variance of the root node degree and hence of the average distance to the root. The distances in the randomized graph were generally lower than in the original Facebook graph due to its lower average path length. Furthermore, $\overline{d}(r)$ correlated more strongly with the RLN, possibly due to the absence of outlier nodes with increased shortest path length. Because of the highly skewed degree distribution of the Ripple graph, the random root node's degree was 1 in 73 out of a 100 runs. The degree of the adversarial node, i.e., 25, was hence generally higher than the degree of the root, leading to shorter paths to the malicious nodes and hence the observed high RLN.

b) Algorithm 1: In addition to $\overline{d}(r)$, we computed the effective average hop distance $D(m) = \overline{d}(m) + d(m, r) - 1$,

as $d(m,r)-1$ is the $level$-value that m propagated during a simulation run. We expected a positive correlation between $\overline{d}(r)-D(m)$ and the RLN, i.e., nodes closer to the root node than $D(m)$ should be well-directed and otherwise not.

In all runs with 25 attack edges, $D(m)$ was considerably higher than $\overline{d}(r)$ such that only a very small number of nodes became ill-directed. Figure 3 thus shows our more distinct results for an adversary with 1000 attack edges, ordered by the RLN. The results indeed validated the expected correlation. Due to the high number of attack edges, the $\overline{d}(m)$ value of each run only differed slightly from its mean value of 2.75, 2.52, 2.63, and 2.02 for the Facebook graph, the randomized Facebook graph, the ER graph, and the Ripple graph, respectively. Thus, the values of $D(m)$ mainly depended on $d(m,r)$ and hence differed by integer values.

Again, the degree of correlation between $\overline{d}(r)-D(m)$ and the RLN varied between graphs. The Facebook graph generally had a longer average shortest path length and hence varied in $\overline{d}(r)$ considerably. In contrast, the value of $\overline{d}(r)$ was more stable for the randomized Facebook graph and the ER graph, so that $d(m,r)$ is indeed the main impact factor.

Here, we also find the explanation for the strong difference between the mean RLN values for the simulations of Algorithm 1 with a cheating adversary and those of Algorithm 1 with a non-cheating adversary on the Ripple graph. It stems from the fact that the $d(m,r)$ value decreased very slowly as the number of attack edges increases. Concretely, the mean value of $\overline{d}(r)$ was roughly 3.8, irrespective of the number of attack edges and the construction algorithm. In the case of 25 attack edges, the mean value for $D(m)$ was 3.9 and in the case of 5000 edges, it was 2.9, such that the $level$ value propagated by m was low enough to cause a high number of nodes to become ill-directed. As the value of $D(m)$ was increased by 1 when the adversary does not cheat, it was higher than $\overline{d}(r)$ for any considered number of attack edges, resulting in a negative $\overline{d}(r)-D(m)-1$ and hence a low impact of the attack. In contrast, $\overline{d}(r)-D(m)$ was positive, corresponding to an attack of high impact.

c) Summary of Results: The first part of our evaluation showed that our protocol based on cryptographic signatures is much more robust to malicious behavior and attacks than state-of-the-art solutions without the usage of cryptography. Indeed, as displayed in Figure 1, to compromise a similar number of nodes, the adversary needs to establish up to 200 times as many attack edges compared to the algorithm by Dubois et al. [6].

VII. Conclusion

In this paper, we leveraged cryptographic signatures to design a BFS tree algorithm that greatly reduces the number of nodes affected by attacks. Based on the concept of topology-aware strict stabilization, we proved that this algorithm only allows malicious nodes to report a distance to the root that differs by at most one from the correct value. Our evaluation based on real-world scenarios demonstrates that this novel construction provides crucial security improvements over existing, non-cryptographic algorithms. Yet, our results indicate that the resistance to attacks is highly correlated with the degree of the root node, highlighting the need to develop secure leader election algorithms that prioritize high-degree nodes.

VIII. Acknowledgements

We thank Sebastién Tixeuil for shepherding our work and the reviewers for their constructive feedback. This work has been funded by the German Research Foundation (DFG) Grant STR 1131/2-2 and EXC 2050 "CeTI´´.

References

[1] Nikita Borisov et al., *Denial of service or denial of security?*, Computer and Communications Security, 2007.
[2] Yazan Boshmaf et al., *The socialbot network: when bots socialize for fame and money*, Computer Security Applications, 2011.
[3] Martin Byrenheid, Stefanie Roos, and Thorsten Strufe, *Attack-resistant spanning tree construction in route-restricted overlay networks*, Tech. report, arXiv preprint 1901.02729, 2019.
[4] Ian Clarke et al., *Private communication through a network of trusted connections: The dark freenet*, https://freenetproject.org/assets/papers/freenet-0.7.5-paper.pdf, 2010.
[5] Danny Dolev and Andrew Yao, *On the security of public key protocols*, Transactions on Information Theory (1983).
[6] Swan Dubois, Toshimitsu Masuzawa, and Sébastien Tixeuil, *Maximum metric spanning tree made byzantine tolerant*, Algorithmica (2015).
[7] Stefan Dziembowski, Sebastian Faust, and Kristina Hostáková, *General state channel networks*, Computer and Communications Security, 2018.
[8] P Erdös and A Rényi, *On random graphs i*, Publicationes Mathematicae Debrecen (1959).
[9] Andreas Hoefer, Stefanie Roos, and Thorsten Strufe, *Greedy Embedding, Routing and Content Addressing for Darknets*, KiVS/NetSys, 2013.
[10] Yih-Chun Hu, David B Johnson, and Adrian Perrig, *Sead: Secure efficient distance vector routing for mobile wireless ad hoc networks*, Ad hoc networks (2003).
[11] Tomas Isdal et al., *Privacy-preserving p2p data sharing with oneswarm*, ACM SIGCOMM Computer Communication Review (2011).
[12] Giulio Malavolta et al., *Silentwhispers: Enforcing security and privacy in credit networks*, Network and Distributed System Security, 2017.
[13] Prateek Mittal, Matthew Caesar, and Nikita Borisov, *X-vine: Secure and pseudonymous routing in dhts using social networks*, Network and Distributed System Security, 2012.
[14] Joseph Poon and Thaddeus Dryja, *The bitcoin lightning network: Scalable off-chain instant payments*, Tech. report, https://lightning.network/lightning-network-paper.pdf, 2016.
[15] Pavel Prihodko et al., *Flare: An approach to routing in lightning network*, 2016, Available at: https://bitfury.com/content/downloads/whitepaper_flare_an_approach_to_routing_in_lightning_network_7_7_2016.pdf.
[16] Stefanie Roos, Martin Beck, and Thorsten Strufe, *Anonymous addresses for efficient and resilient routing in f2f overlays*, INFOCOM, 2016.
[17] Stefanie Roos et al., *Settling payments fast and private: Efficient decentralized routing for path-based transactions*, Networks and Distributed Systems Security, 2018.
[18] Stefanie Roos and Thorsten Strufe, *On the impossibility of efficient self-stabilization in virtual overlays with churn*, INFOCOM, 2015.
[19] Vibhaalakshmi Sivaraman et al., *Routing cryptocurrency with the spider network*, arXiv preprint arXiv:1809.05088 (2018).
[20] Paul F Tsuchiya, *The landmark hierarchy: a new hierarchy for routing in very large networks*, SIGCOMM Computer Communication Review, 1988.
[21] Andras Varga, *OMNeT++ Discrete Event Simulator*, https://omnetpp.org/, Accessed November 2018.
[22] Eugene Vasserman et al., *Membership-concealing overlay networks*, Computer and Communications Security, 2009.
[23] Bimal Viswanath et al., *On the evolution of user interaction in facebook*, Workshop on Online social networks, 2009.
[24] Manel Guerrero Zapata and Nadarajah Asokan, *Securing ad hoc routing protocols*, ACM Workshop on Wireless security, 2002.