Local Checkability in Dynamic Networks

Klaus-Tycho Foerster[*]
Microsoft Research, USA
foklaus@ethz.ch

Oliver Richter
ETH Zurich, Switzerland
richtero@ethz.ch

Jochen Seidel
ETH Zurich, Switzerland
seidelj@ethz.ch

Roger Wattenhofer
ETH Zurich, Switzerland
wattenhofer@ethz.ch

ABSTRACT

In this work we study local checkability of network properties considering inconsistency throughout the verification process. We use disappearing and appearing edges to model inconsistency and prover-verifier-pairs (PVPs) for verification. We say that a network property $\mathcal{N}$ is *locally checkable under inconsistency* if there exists a PVP for a specified inconsistency. In such a PVP, a prover $\mathcal{P}$ assigns a label to each vertex of an initial graph G_i. A distributed time constant verifier $\mathcal{V}$ computes YES on every vertex of the altered graph G_a, if G_a has the property $\mathcal{N}$ and was labeled by $\mathcal{P}$, and NO on at least one vertex of G_a if G_a does not fulfill $\mathcal{N}$.

For s-t-reachability, we present an optimal 2-bit-label PVP considering one disappearing edge and an upper bound of $\mathcal{O}(\log n)$ bits as label size for one appearing edge. For cyclicity, we prove that no general PVP can be found considering disappearing edges, as well as an upper bound of $\mathcal{O}(n^2 \log n)$ bits as the label size for one appearing edge. Furthermore, we show that no PVP can be found for s-t-reachability nor general cyclicity that can consider multiple inconsistencies.

CCS Concepts

•Networks → Network properties; •Computer systems organization → Dependable and fault-tolerant systems and networks; •Theory of computation → Models of computation; •Computing methodologies → Distributed computing methodologies;

Keywords

Local Checking, s-t Reachability, Cyclicity, Dynamic Graphs

1. INTRODUCTION

A network administrator must know whether the network is correct [20]. E.g., is destination t reachable from source s, or may the current forwarding rules send packets in a cycle?

[*]Most of the work was done while being at ETH Zurich.

ICDCN '17, January 04-07, 2017, Hyderabad, India
© 2017 ACM. ISBN 978-1-4503-4839-3/17/01…$15.00
DOI: http://dx.doi.org/10.1145/3007748.3007779

Edge Inconsistencies	s-t reachability	Cyclicity (max. 1 cycle)	Cyclicity (general)
One Disappearing	2	2	NLC
One Appearing	$\mathcal{O}(\log n)$	$\mathcal{O}(n^2 \log n)^+$	$\mathcal{O}(n^2 \log n)^+$
Multiple Disappearing	NLC	2	NLC
Multiple Appearing	NLC	NLC	NLC
Combinations	NLC	NLC	NLC

Table 1: Main findings of this work summarized in a table. The table entries show the proof label size (in bits) of a prover-verifier algorithm solving the problem or NLC if the problem is Not Locally Checkable, i.e., not solvable in the model we introduce. Two entries carry the addition "+", which means that these algorithms are only feasible if the graph vertices carry unique identification tags (IDs). Reachability is covered in Section 3, Cyclicity in Section 4.

Traditionally, such network properties are checked by sending probing packets into the network. It is potentially much less costly to check these *global* network properties by inexpensive *local* verification, e.g. [8, 19]: each node just compares its state with the state of its neighbors, and raises an alarm if it senses that something is not right. Such local checkability is studied and well understood.

In real networks, however, links appear and disappear regularly. Unfortunately, previous work on local checkability does not support such dynamic situations. In this work we explore the possibilities and limitations of local checkability of global network properties in *dynamic* networks. We study disappearing and/or appearing edges. E.g., is there a way to locally determine whether destination t is still reachable from source s even after an arbitrary link went down?

We check a property through a distributed nondeterministic algorithm modeled as *prover-verifier-pair* (PVP) as introduced in [8]: A *prover* $\mathcal{P}$ assigns specific labels to all vertices (the *proof*) if the graph has (or might get through inconsistencies) the property in question. From then on, a distributed deterministic *verifier* $\mathcal{V}$ can verify the property by taking as input on each vertex v only the label of v and its immediate neighbors' labels.

The results we found for the problems addressed are summarized in Table 1. Besides these results the contributions of this work are a prover-verifier-pair for checking s-t reachability if one edge in the graph might disappear or appear between labeling and verification; and the fact that no such pair can be found for checking cyclicity in a graph where one edge might disappear without losing the generality that a graph might have multiple cycles.

Due to the space restrictions for this extended abstract, some proofs are deferred to the full version of this article.

2. DEFINITIONS AND MODEL

Networks are modeled as graphs $G = (V(G), E(G))$ in this article, where $V(G)$ denotes the set of vertices and $E(G)$ the set of edges. If the graph G is clear from the context, a shorter notation $V = V(G)$ and $E = E(G)$ is used. This work focuses on *undirected graphs*, i.e., if $u, v \in V(G)$ then $e = (u, v) \in E(G)$ is equivalent to $e' = (v, u)$ or in short $e = e' = \{u, v\}$. For any vertex $v \in V$ we denote the number of edges adjacent to v by $\deg(v)$, called *degree of v*.

The function $l : V \to \{0, 1\}^*$ shall denote a *(vertex) labeling* for G that assigns a finite bit-string as *label* to each vertex in V. Given a graph G with labeling l, let $M(v) = \{l(u_1), ..., l(u_{deg(v)})\}$ denote the *set of messages* vertex v receives in one communication round, where $u_1, ..., u_{\deg(v)} \in V$ are the immediate neighbors of v. Note that the set $M(v)$ is unordered and neither vertex identifiers nor port numbers are attached to it. The function $o_{M(v)} : \{0, 1\}^* \to \mathbb{N}_0$ shall denote the number of occurrences a given label has in the set of messages $M(v)$. If $M(v)$ is clear from the context we will use the short form $o(\cdot)$. Furthermore, in this article logarithms are rounded up to integer value and use base 2.

To model inconsistency, G_i shall denote the *initial graph*, the graph in question before any inconsistencies are considered, and G_a shall denote the *altered graph*, the graph after all inconsistencies were considered. Since we only alter graph edges to model inconsistency $V(G_i) = V(G_a) = V$. Note that even though the set of vertices V stays the same, the degree $\deg(v)$ and set of messages $M(v)$ for a given $v \in V$ might change. However, we shall evaluate $\deg(v)$ and $M(v)$ solely on the altered graph G_a.

2.1 Inconsistencies

Inconsistency is modeled through *disappearing edges* and *appearing edges*. For $u, v \in V$, a *disappearing edge* $d = \{u, v\} \in E(G_i)$ is an edge that is part of the initial graph G_i but not of the altered graph, i.e., $d \notin E(G_a)$. For $x, y \in V$ an *appearing edge* $a = \{x, y\} \in E(G_a)$ is an edge that is part of the altered graph G_a but was not in the initial graph, i.e., $a \notin E(G_i)$. Thus, $E(G_a) = E(G_i)\backslash D \cup A$, where D denotes the set of all disappearing edges and A denotes the set of all appearing edges.

In our model, any vertex $v \in V(G_a)$ gets an additional 3 bits of information $R(v) = \{r_1, r_2, r_3\}$ to report inconsistency: r_1 is set if and only if v has an adjacent disappearing edge, r_2 is set if and only if v has an adjacent appearing edge, and r_3 is set if and only if there has been an appearing edge, i.e., if $A \neq \emptyset$. We will refer to these bits as *first, second,* or *third inconsistency reporting bit*, respectively.

2.2 Prover-Verifier-Pair

A *network* or *graph property* is defined as a set Y containing all graphs that posses a common attribute, also referred to as YES-*instances*. Any graph $G \notin Y$ is referred to as NO-*instance*. We verify graph properties in two steps, a *proof step* and a *verification step*.

In the *proof step*, a *prover* $\mathcal{P}$ gets as input an initial graph G_i and computes for every vertex $v \in V(G_i)$ a finite label $l(v)$. We refer to this labeling l as the *proof*.

For the *verification step* let G_a be any graph and let l be any labeling for G_a. A *verifier* $\mathcal{V}$ is a distributed algorithm that gets as input on every vertex v: the vertex label $l(v)$, the set of messages $M(v)$ containing the labels of neighboring vertices, and the three inconsistency reporting bits; and computes as output for each vertex either YES or NO.

A *prover-verifier-pair* $(\mathcal{P}, \mathcal{V})$ or PVP in short is said to be *correct for* Y if the following two points hold: First, if $G_a \in Y$ and l was obtained from $\mathcal{P}$, then $\mathcal{V}$ computes YES as output for all vertices, and second, if $G_a \notin Y$, then $\mathcal{V}$ computes NO as output for at least one vertex, regardless of the labeling.

A network property Y is *not locally checkable* (or NLC in short) if there exists no PVP that is correct for Y. To measure the quality of a PVP we specify the *proof size* $f(n)$ as the maximal number of bits any vertex of any YES-instance with n vertices might get assigned by the prover. For a given network property Y, the smallest proof size for which there exists a correct PVP for Y defines the *proof size for* Y.

3. REACHABILITY

One of the fundamental questions in network theory is the question of *s-t-reachability*: Can a given source vertex $s \in V$ in the network reach a destination vertex $t \in V$ in the network? This question is the basis for all data transfer throughout the network since two devices (network vertices) can only communicate if they can reach each other.

More formally: $s \in V$ can *reach* $t \in V$ if and only if there exists a path from s to t in the underlying network graph.

For two vertices $u, v \in V$ for which there exists a path from u to v we will use the two expressions "u can *reach* v" and "u is *connected* to v" interchangeably.

To make meaningful statements about reachability or connectivity we focus in this section on finite graphs with at least two vertices from which one is carrying the unique label s and another is carrying the unique label t, cf. [14].

3.1 One Disappearing Edge

First, we take a look at what can happen if at most one edge d disappears, i.e., $E(G_a) = E(G_i) \vee E(G_i)\backslash\{d\}$, where d can be any edge of the initial graph.

Here we focus on the case that s-t-reachability is given in the initial graph since this is the case where the property might get altered. Therefore let s-t-REACHABILITY denote the set of all undirected graphs with a path from s to t.

In this subsection we first explain the proof idea and then prove the following theorem:

THEOREM 1. *The proof size for* s-t-REACHABILITY *is 2 bit if at most one edge is disappearing.*

We look for a prover-verifier-pair that takes the assumption of a disappearing edge into consideration. Considering s-t-reachability given initially, the property only changes if the disappearing edge d is on the path from s to t and there exists no other path from s to t that does not include d. Therefore, a prover considering one disappearing edge should mark two distinct paths wherever possible to still prove the property even if one of these paths breaks due to the disappearing edge. With *distinct paths* we thereby refer to paths that do not share any vertices except for the first and last.

More precisely, any s-t-path P can be segmented in *critical path segments* and *uncritical path segments*, where *critical path segments* are path segments that are also part of all other paths from s to t and *uncritical path segments* are the remaining segments of P. A prover can now differentiate between *critical-path-vertices*, *uncritical-path-vertices*

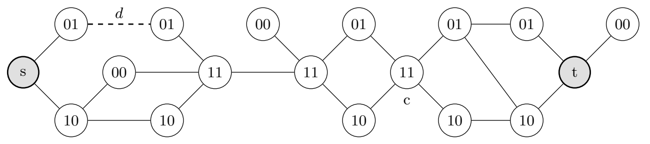


Figure 1: Example graph showing a possible 2 bit proof labeling to verify s-t-reachability considering that at most one edge might disappear, e.g., d. Here, non-path-vertices are labeled 00, critical-path-vertices are labeled 11 and uncritical-path-vertices 10 or 01, respectively, marking the two possible uncritical path segments chosen by the prover. Note that a critical path segment can consist of a single vertex.

and *non-path-vertices*. Thereby, *critical-path-vertices* are all vertices that are on every path from s to t, i.e., all vertices on critical path segments including the endpoints of the critical path segments. *Uncritical-path-vertices* are vertices chosen by the prover to mark two distinct paths for every uncritical path segment, whereas *non-path-vertices* are the remaining vertices in the graph. A simple coding of these vertex classes generates the labeling the prover can put on the graph $G_i \in Y$.

A verifier can then read neighboring labels together with the first inconsistency reporting bit on every vertex $v \in V$ of the altered graph G_a and verify

- if the proof-labeling is a correct proof-labeling indicating s-t-reachability before the graph was altered by checking if the amount of each label class in $M(v)$ aligns with a possible proof labeling of a Yes-instance
- if the disappearing edge (if there is any) disappeared on a critical path segment, on an uncritical path segment or not on any path specified by the prover by checking the vertex label $l(v)$ and the first inconsistency reporting bit on the vertex
- if s-t-reachability is given in the altered graph by taking the first two points into consideration

Consider Figure 1 as a possible example how such a proof labeling might look like.

To prove Theorem 1 we first consider the following lemma:

Lemma 2. *There exists a correct PVP with a proof size of 2 bit that is correct for s-t-*reachability *if at most one edge d disappears.*

Proof. A prover-verifier-pair $(\mathcal{P}, \mathcal{V})$ as required is described here. Given $G_i = (V, E_i) \in s$-t-reachability let $C \subset V$ denote the set of all critical path vertices. Note that $s, t \in C$.

The prover $\mathcal{P}$ first labels all vertices $v_c \in C \backslash \{s, t\}$ with the label $l(v_c) = 11$. For a given path from s to t, each uncritical path segment U_i has a starting vertex $u_s(U_i) \in C$ and an termination vertex $u_t(U_i) \in C$. For each uncritical path segment U_i the prover $\mathcal{P}$ chooses two distinct paths $P_1(U_i)$, $P_2(U_i)$ from $u_s(U_i)$ to $u_t(U_i)$ and labels all not yet labeled vertices on $P_1(U_i)$ with 01 and all not yet labeled vertices on $P_2(U_i)$ with 10. Such two distinct paths for uncritical path segments exist as a consequence of the definition of critical-path-vertices. $\mathcal{P}$ then labels all remaining vertices with the label 00. See Figure 1 as an example of this proof labeling.

For the verifier $\mathcal{V}$ consider the pseudo code in Algorithm 1.

Given $G_a \in s$-t-reachability we show that $\mathcal{V}$ outputs Yes on all vertices if l was obtained from $\mathcal{P}$:

- All vertices labeled 00 output Yes by default
- All vertices labeled 01 or 10 are on an uncritical path segment. Therefore they have exactly two neighbors indicating a possible s-t-path or one such neighbor and an adjacent edge that disappeared. Note that for the latter s-t-reachability is still given since the one and only disappearing edge disappeared on an uncritical path segment. In both cases the verifier $\mathcal{V}$ will output Yes.
- All vertices labeled 11 are critical-path-vertices that have either

  1. two critical-path-vertices as neighbors
  2. one critical-path-vertex as neighbor and they are an end point of an uncritical path segment or
  3. no critical-path-vertex as neighbor, meaning that they are in between two uncritical path segments (see vertex c in Figure 1 as an example)

  If no adjacent edge disappeared this can be summarized to $y = 2$, where y is the number of critical-path-vertices in $M(v)$ plus the number of uncritical-path-vertices pairs. This holds since for every adjacent uncritical path segment there must be one neighbor labeled 01 and either one neighbor labeled 10, forming a pair, or directly another critical vertex. Furthermore the path should continue in both directions (as critical or uncritical path segment) which leads to $y = 2$.
  If however an edge disappeared on an adjacent uncritical path, point 2 and 3 of the enumeration above have to be accounted for and are summarized to $\{y = 1 \wedge o(01) + o(10) = 1 \wedge r_1 \textit{ is set}\}$ and $\{y = 1 \wedge o(01) + o(10) = 3 \wedge r_1 \textit{ is set}\}$ respectively.
  In all of these cases the verifier $\mathcal{V}$ will output Yes.
- Finally, a vertex carrying the label s or t marks the start or end of the path in question, therefore it neighbors either a critical-path-vertex or an uncritical path segment. With the same argumentation as before this can be summarized to $y = 1$ if no adjacent edge disappears or $y = 0 \wedge o(01) + o(10) = 1 \wedge r_1 \textit{is set}$ if an adjacent uncritical path edge disappears. In both cases the verifier $\mathcal{V}$ will output Yes.

Therefore $\mathcal{V}$ will generate the output Yes on all vertices of $G_a \in s$-t-reachability if l was obtained from $\mathcal{P}$.

What is left to show is that given $G_a \notin s$-t-reachability, $\mathcal{V}$ will compute No as output on at least one vertex.

Consider for the sake of contradiction that there exists a graph $G_a \notin s$-t-reachability that triggers $\mathcal{V}$ to compute Yes on every vertex.

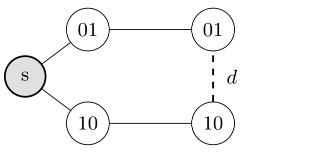
Such a graph must have a vertex s due to the problem setting. Since $o(l) \geq 0$ for all labelings l, Algorithm 1 requires s to have either

- Case 1: a vertex labeled as critical-path-vertex as neighbor. In this case a vertex labeled 11 since the only other possibility (t) would lead already to a contradiction.
- Case 2: a vertex labeled 01 and a vertex labeled 10 as neighbor. Or
- Case 3: a vertex labeled 01 or 10 as neighbor and an adjacent disappearing edge (r_1 set)

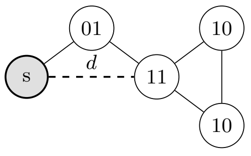
We consider these cases separately:

Algorithm 1: Pseudo code of verifier $\mathcal{V}$ checking s-t-reachability if at most one edge is disappearing. G_a denotes the altered graph, $l(v)$ the label on vertex v, $M(v)$ the set of messages received by v, $r_1(v)$ the first inconsistency reporting bit (reporting an adjacent disappearing edge) and $o(\cdot)$ the number of occurences a given label has in $M(v)$.

```
foreach $v \in V(G_a)$ do
    Input: $l(v), M(v), r_1(v) \in R(v)$
    $o(\cdot) \equiv o_{M(v)}(\cdot)$
    if $l(v) = 00$ then
        Output: YES
    if $l(v) = 01$ OR $l(v) = 10$ then
        // Define $x$ as the number of path vertices in the neighborhood
        $x = o(11) + o(s) + o(t) + o(l(v))$
        if $x = 2$ OR $x = 1 \wedge r_1$ is set then
            Output: YES
        else
            Output: NO
    // Set $y$ as the number of critical-path-vertices in the neighborhood
    $y = o(11) + o(s) + o(t)$
    // A pair$\{01, 10\}$ is a continuation of the path
    foreach pair $\{01, 10\} \in M(v)$ do
        $y = y + 1$
    if $l(v) = 11$ then
        if $y = 2$ OR
        $y = 1 \wedge o(01) + o(10) = 1 \wedge r_1$ is set OR
        $y = 1 \wedge o(01) + o(10) = 3 \wedge r_1$ is set then
            Output: YES
        else
            Output: NO
    if $l(v) \in \{s, t\}$ then
        if $y = 1$ OR
        $y = 0 \wedge o(01) + o(10) = 1 \wedge r_1$ is set then
            Output: YES
        else
            Output: NO
    if $l(v) \notin \{s, t, 00, 01, 10, 11\}$ then
        Output: NO
```



(a) Graph with disappearing edge d connecting two different uncritical paths into a loop



(b) Graph with disappearing edge d leading to a loop after a critical-path-vertex.

Figure 2: Example graphs to show that one side might end in a loop. Here d denotes a disappearing edge. In these graphs, the verifier $\mathcal{V}$ would compute YES for every vertex. However these graphs do not contain t which contradicts the model assumption. Since there is at most one edge disappearing, any subgraph containing t must be connected to s or grow to infinity if it had to output YES on all vertices as well.

Case 1: For the vertex v labeled 11, $y \geq 1$ since a critical vertex (s) is its neighbor. Therefore Algorithm 1 requires either

- $y = 2$ which requires a next vertex to have a label 11 having v as neighbor (which is equivalent to case 1) or a pair$\{01, 10\}$ as next neighbors(which is equivalent to case 2). Or
- $y = 1$ and a vertex labeled 01 or 10 as additional neighbor and an adjacent disappearing edge. This is equivalent to case 3.

Case 2: A vertex u carrying a label $l \in \{01, 10\}$ outputs YES if

- two of its neighbors report to be either a critical-path-vertex or to have the same label as u has. This ensures a continuation of the uncritical path up to a next critical-path-vertex, considering that G_a is finite. This leads to case 4. Or
- only one neighbor reports to be critical-path-vertex or to have the same label as u and u has an adjacent disappearing edge. This can happen to one of the two paths initiated by the first two vertices labeled 01 and 10 or the disappearing edge disappeared between a vertex labeled 01 and a vertex labeled 10. If only one path gets interrupted by a disappearing edge, the other path is guaranteed to lead to a next critical-path-vertex (case 4). If the disappearing edge connects both path ends (see Figure 2a for illustration), we have a loop requiring a disappearing edge (case 5).

Case 3: Similar to case 2 here the disappearing edge is already at the start of one of the two uncritical paths. Since there is at most one disappearing edge, Algorithm 1 ensures a continuation of the other path to a next critical-path-vertex (case 4) or that the path ends up on the other side of the edge that disappeared, forming a loop (case 5).

Case 4: In this case we arrive at a critical-path-vertex v_c that must carry the label 11 (since t is not an option) and has at least one neighbor carrying a label $l \in \{01, 10\}$. Algorithm 1 requires here for a YES either

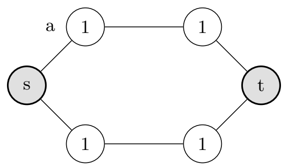
- $y = 1$ leading to a next critical-path-vertex (case 1),
- $y = 2$ from two pairs$\{01, 10\}$ leading to two next uncritical path vertices with distinct labels (case 2) or
- $o(01) + o(10) = 3 \wedge r_1$ *is set*. If the two next path neighbors have distinct labels, we are again at case 2. However if they have the same label, this is a case where the path might end in a loop (see Figure 2b for illustration). Note that this case requires an adjacent disappearing edge. Therefore we are again at case 5.

Case 5: In this case we end up in a loop that somewhere required a disappearing edge (see Figure 2 for two examples). Since there is at most one disappearing edge and the whole path argumentation can also be started from t instead of s, only one side can end in this case.

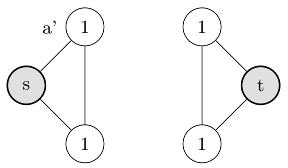
Therefore, since every case leads to another case, at least from one side the path continues to infinity which contradicts the assumption of a finite graph G_a.

This concludes the proof of Lemma 2. □

REMARK 3. *Note that this PVP does not deliver the correct output for each individual connected component separately, since on one side the path might end up in a loop*

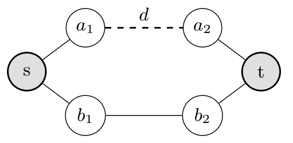


(a) Graph with s-t-reachability

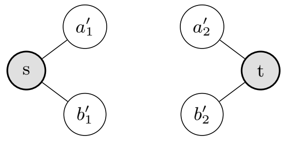


(b) Graph where s-t-reachability is not given

Figure 3: Example graphs showing that extending the idea of marking two paths from s to t with the same label each will not work: The vertices a and a' get exactly the same information, therefore it is indistinguishable for a verifier whether the vertices are on a loop or an s-t-path.



(a) Graph with s-t-reachability and a disappearing edge d



(b) Graph without s-t-reachability

Figure 4: Example graphs to show why the ability to sense an adjacent disappearing edge is necessary. If a vertex cannot sense an adjacent disappearing edge, vertices a_1', a_2', b_1' and b_2' would all get the same information as a_1 and a_2 and therefore would be indistinguishable for a verifier.

as shown in Figure 2. Interestingly, we can show that no such PVP exists that can consider each individual connected component separately. The proof details are deferred to the full version of this article.

LEMMA 4. *The proof size for s-t-*REACHABILITY *considering that one edge d might disappear is at least 2 bits.*

Combining Lemma 2 and Lemma 4 yields Theorem 1. Again, due to space restrictions, the proof details of Lemma 4 are deferred to the full version of this article. We refer to Figure 3 for an example why extending the standard checkability idea without failing edges does not suffice.

3.2 One Appearing Edge

In this subsection we look at what happens if at most one edge appears, i.e., $E(G_a) = E(G_i) \vee E(G_i) \cup \{a\}$ where a can appear between any two vertices $u, v \in V$. We focus on the non-trivial case of s-t-reachability not given initially, first explaining the idea and then proving the following theorem:

THEOREM 5. *There exists a PVP with a proof size in $\mathcal{O}(\log n)$ bit that is correct for s-t-*REACHABILITY *if at most one edge a appears.*

If s-t-reachability is not given initially, s and t must be located in two different connected components of the graph. An appearing edge a can make the altered graph $G_a \in s$-t-REACHABILITY if and only if it connects these two connected components. Since a can appear between any two vertices, every vertex must be able to verify locally

1. if it is connected to s, t or neither s nor t

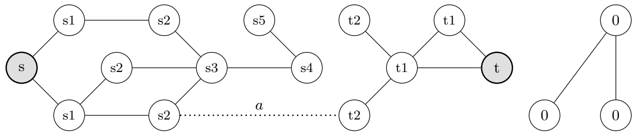


Figure 5: Example graph showing a possible $\mathcal{O}(\log n)$ bit proof labeling to verify s-t-reachability considering that at most one edge might appear, e.g., a. Here, proof labels consist of s or t and the distance to s or t. For vertices not connected to neither s nor t the label 0 is used.

2. if it got an adjacent appearing edge connecting it to a different connected component, possibly resulting in s-t-reachability
3. if an edge appeared somewhere else in the graph, possibly resulting in s-t-reachability

The second and third point can be checked by the second and third inconsistency reporting bits.

The first point can be assured by introducing a labeling composed of a *prefix* and a *distance*. Thereby the *prefix* of each label $l(v)$ indicates whether v is connected to s or to t and the *distance* indicates the minimal distance to s or t, respectively. Since we focus on the non-trivial case, a vertex cannot be connected to both s and t at the time of labeling. If a vertex is connected to neither s nor t, a simple label 0 can be used.

To decode the prefix of such a labeling we introduce a *prefix function* $p : \{0,1\}^* \to \{s, t, 0\}$ that takes a label l_v and computes whether the label contains a prefix referring to s ($p(l_v) = s$), a prefix referring to t ($p(l_v) = t$) or no prefix ($p(l_v) = 0$). Note that $p(s) = s$ and $p(t) = t$.

To decode the distance of such a labeling we introduce a *distance function* $d : \{0,1\}^* \to \mathbb{N}_0 \cup \{\infty\}$ that takes a label l_v and returns

- 0, if $l_v \in \{s, t\}$
- the distance, if l_v contains a distance > 0 or
- ∞ if l_v does not contain a distance or an invalid (e.g., ≤ 0) distance and l_v is neither s nor t.

PROOF. For $G_i = (V, E_i) \notin s$-t-REACHABILITY we present a PVP $(\mathcal{P}, \mathcal{V})$ as required. Let $S \subset V$ denote the set of all vertices v_s that s can reach, i.e., for which there exists a path from s to v_s. Similarly let $T \subset V$ denote the set of all vertices that can be reached by t.

The prover $\mathcal{P}$ labels all vertices $v_s \in S$ with the label s concatenated with the shortest distance to s. Similarly $\mathcal{P}$ labels all $v_t \in T$ with t and the shortest distance to t. All remaining vertices $v \in V \backslash \{S \cup T\}$ are labeled with the label 0. Since any distance is strictly smaller than the number of vertices n and the labels s and t are of constant size, this proof size is in $\mathcal{O}(\log n)$. An illustration of such a labeling can be found in Figure 5.

For the verifier $\mathcal{V}$ consider the pseudo code in Algorithm 2.

Given $G_a \in s$-t-REACHABILITY we show that $\mathcal{V}$ outputs YES on all vertices if l was obtained from $\mathcal{P}$. Note that in this case an edge a must have appeared connecting the two connected components containing s and t, respectively. Therefore the third inconsistency reporting bit $r_3(v) \in R(v)$ is set for all vertices $v \in V$. Looking at Algorithm 2 and given that the labeling l was obtained from $\mathcal{P}$ we note that

- each vertex v_0 labeled $l(v_0) = 0$ outputs YES since

Algorithm 2: Pseudo code of verifier $\mathcal{V}$ checking s-t-reachability if at most one edge is appearing. G_a denotes the altered graph, $l(v)$ the label on vertex v, $M(v)$ the set of messages received by v, $r_2(v)$ the second inconsistency reporting bit (reporting an adjacent appearing edge) and $r_3(v)$ the third inconsistency reporting bit (reporting an appearing edge in the graph). $o_{prefix}(k)$ shall denote the number of labels in $M(v)$ carrying the prefix k, where $k \in \{s,t\}$. $p(\cdot)$ denotes the prefix decoding function and $d(\cdot)$ denotes the distance decoding function.

```
foreach $v \in V(G_a)$ do
    Input: $l(v), M(v), r_2(v) \in R(v), r_3(v) \in R(v)$
    if $r_3(v)$ is not set then
        Output: No
    if $l(v) = 0$ then
        if $r_2(v)$ is set then
            Output: No
        else
            Output: Yes
    // Decode $l(v)$ into prefix $p_v \in \{s,t\}$ and distance $d_v$.
    $p_v = p(l(v))$
    $d_v = d(l(v))$
    if $p_v = 0$ OR $d_v = \infty$ then
        Output: No
    if $l(v) \in \{s,t\} \wedge (s \in M(v)$ OR $t \in M(v))$ then
        Output: Yes
    if $d_v = 0 \wedge \nexists l_0 \in M(v) : d(l_0) = 0$ OR
      $\exists l_1 \in M(v) : d(l_1) = d_v - 1$ then
        if $r_2(v)$ is set then
            if $o_{prefix}(p_v) = |M(v)| - 1 \wedge$
              $\exists!\ l \in M(v) : p(l) \in \{s,t\} \wedge p(l) \neq p_v$ then
                Output: Yes
        else
            if $o_{prefix}(p_v) = |M(v)|$ then
                Output: Yes
    if non of the cases above then
        Output: No
```

the the appearing edge a connects the connected component containing s to the connected component containing t and therefore no vertex labeled 0 can have an adjacent appearing edge

- vertices s and t have a distance $d_v = 0$ and no neighbor with distance 0 unless the edge appeared between s and t (which is separately handled in the algorithm).
- each vertex $v \in \{S \cup T\} \backslash \{s,t\}$ has at least one neighbor with distance $d_n = d_v - 1$ where d_v denotes the distance of v
- there exists a vertex $v_{a,s} \in S$ and a vertex $v_{a,t} \in T$ that have an adjacent appearing edge (second inconsistency reporting bit $r_2(v) \in R(v)$ set). $v_{a,s}$ and $v_{a,t}$ receive exactly one prefix $p(l) \in \{s,t\} \neq p_v$ where p_v denotes their own prefix. This prefix $p(l)$ is the one received through the edge a that appeared. All other prefixes received from neighbors are equal to p_v since they were all in the same connected component of the initial graph G_i. Given all this, $\mathcal{V}$ computes Yes for $v_{a,s}$ and $v_{a,t}$
- for each vertex $v \in \{S \cup T\} \backslash \{v_{a,s}, v_{a,t}\}$ $r_2(v)$ is not set, since there is at most one appearing edge. All prefixes received by v are equal to its own prefix p_v since they were all in the same connected component of the initial graph G_i. Therefore $\mathcal{V}$ will compute Yes for all those vertices as well

This concludes that $\mathcal{V}$ will compute Yes for every vertex of an altered graph $G_a \in s$-t-REACHABILITY if the labeling l was obtained from $\mathcal{P}$.

What is left to show is that given $G_a \notin s$-t-REACHABILITY, $\mathcal{V}$ will compute No as output on at least one vertex.

Assume for the sake of contradiction that there exists a graph $G_a \notin s$-t-REACHABILITY for which $\mathcal{V}$ computes Yes on all vertices.

Considering Algorithm 2 and the case of no edge appearing ($r_3(v)$ is not set for any $v \in V$), $\mathcal{V}$ would compute No for all vertices. Therefore G_a must have an edge a that appeared.

We further note that for any vertex v carrying a prefix p_v the following must hold:

1. if $r_2(v)$ is not set, all neighbors of v must carry the same prefix p_v since $\mathcal{V}$ requires $o_{prefix}(p_v) = |M(v)|$ to output Yes
2. if $r_2(v)$ is set, all neighbors but one must carry the same prefix p_v and the remaining neighbor must have a label l with prefix $p(l) \in \{s,t\}$ not equal to p_v. Since there are only the prefixes s and t, $p(l) = t$ if $p_v = s$ and vice versa.

From Point 1 it follows that if any vertex within a connected component carries a prefix p_v and no vertex within that connected component has an adjacent appearing edge, all vertices of the connected component must carry the prefix p_v. From Point 2 it follows that if a vertex v has an adjacent appearing edge, v must be connected to a vertex carrying a different prefix $p(l) \in \{s,t\}$. Thus, if the appearing edge a is adjacent to any vertex carrying a prefix, there must be a connection between a connected component carrying only prefixes s and a connected component with only prefixes t.

Next, consider that Algorithm 2 requires every vertex v with $l(v) \neq 0$ to have a prefix $p_v \in \{s,t\}$ and a distance $d_v \neq \infty$. We note that distances assigned by the labeling $l(\cdot)$ must be strictly > 0 for vertices $v \notin \{s,t\}$ since otherwise $d(\cdot)$ would decode them as ∞ which would lead $\mathcal{V}$ to output No. Furthermore, each vertex v with $d_v > 0$ requires a neighbor u with distance $d_u = d_v - 1$, especially, vertices having a distance 1 require either s or t in their neighborhood.

Therefore, since s and t are unique, there can only be one connected component carrying the prefixes s containing s itself and one connected component carrying the prefixes t containing t itself. None of those can have the appearing edge a adjacent to any of their vertices since this would, as shown earlier, request a connection between the two components, thereby connecting s and t.

The only accepted label not carrying a prefix is 0. However, if an appearing edge a were adjacent to a vertex v carrying the label $l(v) = 0$, $\mathcal{V}$ would compute No as output for v. Therefore, since there has to be an appearing edge a but no vertex in G_a can have an adjacent appearing edge, we have a contradiction proving our initial claim. □

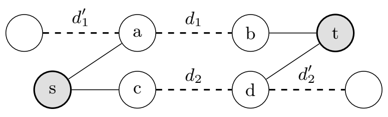


Figure 6: Graph showing that s-t-reachability cannot be verified in a local manner if there might be two edges disappearing. Here, vertex a can sense an adjacent disappearing edge d_1, but cannot receive enough information within one round of communication to decide if a second disappearing edge disappeared at a location d_2 or d_2'. Since it might be d_2', a verifier on a has to output YES. The same applies however to b as well as to c and d, where c and d cannot know whether d_1 or d_1' disappeared.This leads all vertices to output YES even though s-t-reachability might not be given.

3.3 Multiple Inconsistencies

In this subsection we present the challenges in checking s-t-reachability when considering multiple inconsistencies. We will first look at multiple disappearing edges, then look at multiple appearing edges, and end with combinations of disappearing and appearing edges. Due to space constraints, the technical proofs of this subsection are deferred to the full version of this article.

3.3.1 Multiple Disappearing Edges

We consider the initial graph $G_i \in s$-t-REACHABILITY and ask whether there is a correct PVP to verify if $G_a \in s$-t-REACHABILITY with $E(G_a) = E(G_i)\backslash D$ where D denotes the set of disappearing edges. Sadly, the answer is no, since edges might disappear anywhere in the graph and the verifier on a vertex v can only check the local neighborhood of v. We refer to Figure 6 for an illustration of the deferred proof.

THEOREM 6. *There exists no correct PVP for s-t-REACHABILITY as specified in our model that can consider more than one disappearing edge.*

3.3.2 Multiple Appearing Edges

We consider the initial graph $G_i \notin s$-t-REACHABILITY and ask whether there is a correct PVP to verify if $G_a \in s$-t-REACHABILITY with $E(G_a) = E(G_i) \cup A$ where A denotes the set of appearing edges.

As with disappearing edges in Section 3.3.1 appearing edges can appear anywhere in the graph and therefore cannot be located within the graph by a non-adjacent vertex. We refer to Figure 7 for an illustration of the deferred proof.

THEOREM 7. *There exists no correct PVP for s-t-REACHABILITY as specified in our model that can consider more than one appearing edge.*

3.3.3 Combination of Dis– and Appearing Edges

We consider now any initial graph G_i and ask whether there is a correct PVP to verify if $G_a \in s$-t-REACHABILITY with $E(G_a) = \{E(G_i) \cup A\}\backslash D$ where A denotes the set of appearing edges and D denotes the set of disappearing edges.

Assuming that there is at most either one edge disappearing or at most one edge appearing, a combination of the algorithms presented in Section 3.1 and Section 3.2 could make up a PVP for s-t-REACHABILITY with proof size in $\mathcal{O}(\log n)$. This by choosing the proof labeling according to whether the initial graph G_i is in s-t-REACHABILITY or not (if it is, choose the labeling from Section 3.1, if not choose the labeling form Section 3.2) and adapt the verifier to switch to Algorithm 1 or Algorithm 2 depending on the proof labeling it finds.

However, assuming that there is at most one edge disappearing AND at most one edge appearing, a correct PVP for s-t-REACHABILITY cannot be found. This due to the fact that if there is a disappearing edge d breaking the graph into two connected components, vertices adjacent to d cannot know if there is an appearing edge on the other end of the graph connecting the two components again. We refer to Figure 8 for an illustration of the deferred proof.

THEOREM 8. *There exists no correct PVP for s-t-REACHABILITY as specified in our model that can consider a combination of at most one disappearing edge and of at most one appearing edge.*

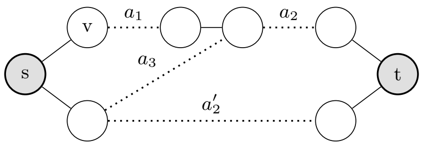


Figure 7: Graph showing that s-t-reachability cannot be verified in a local manner if there might be two edges appearing. Here, vertex v can sense an adjacent appearing edge a_1, but cannot receive enough information within one round of communication to decide if a second appearing edge a_2 or a_2' exists that would lead to s-t-reachability. Note that even if v would get the information of another appearing edge, it could also be that the second edge that appeared is a_3 (and not a_2 or a_2'), leaving s and t unconnected.

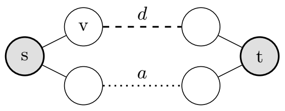


Figure 8: Graph showing that s-t-reachability cannot be verified in a local manner if there might be an edge disappearing and an edge appearing. Here, vertex v can sense an adjacent disappearing edge d, but cannot receive enough information within one round of communication to decide if an appearing edge a on the other side of the graph exists that would lead to s-t-reachability.

4. CYCLICITY

Cyclicity, i.e., if a given graph contains edges that form a cycle, is one of the key attributes when analyzing networks. Especially for routing protocols it is important to know if the routed packet gets forwarded indefinitely in a cycle or is guaranteed to advance to a new vertex with every new step.

In the following subsections we discuss if a prover verifier pair can be found to check if a graph contains a cycle or not under the assumption of inconsistency. We will look at individual connected components of graphs. Therefore we introduce $C(G) = (V_C, E_C)$ to be a *connected component* of a graph G, with $V_C \subset V(G)$ and $E_C \subset E(G)$ such that any two vertices $u, v \in V_C$ can reach each other. We further introduce two small changes to our model:

1. Let Y be a network property. A prover verifier pair $(\mathcal{P}, \mathcal{V})$ is said to be *correct for* Y if the following two points hold for any connected component $C(G_a) = (V_{C,a}, E_{C,a})$ of the altered graph G_a:
   (a) if $C(G_a) \in Y$ and l was obtained from $\mathcal{P}$, then $\mathcal{V}$ computes YES as output for all vertices $v \in V_{C,a}$
   (b) if $C(G_a) \notin Y$, then $\mathcal{V}$ computes NO as output for at least one vertex $v \in V_{C,a}$, regardless of the labeling
2. for a vertex $v \in V(G_a)$, the third inconsistency reporting bit $r_3(v)$ is set if and only if there has been an appearing edge adjacent to v or any vertex u connected to v, i.e., if there has been an appearing edge adjacent to any vertex of the connected component $C(G_a)$ to which $v \in V_{C,a}$ belongs

1-CYCLIC we shall denote the set of all (sub-)graphs containing exactly one cycle, and CYCLIC shall denote the set of all (sub-)graphs containing at least one cycle.

4.1 Disappearing Edges

We first look at the case that edges might disappear. As it was with *s*-*t*-reachability in Section 3.1 we focus here on the case that the initial graph $G_i \in$ CYCLIC.

We divide this problem further into two cases:

1. The initial graph G_i contains exactly one cycle, i.e., $G_i \in$ 1-CYCLIC
2. The initial graph G_i contains at least one cycle, i.e., $G_i \in$ CYCLIC

4.1.1 Single Cycle

In [8] a proof size of 2 bits was shown for cyclicity checking in unaltered undirected graphs. For the case that $G_i \in$ 1-CYCLIC has at most one cycle, the PVP [8] introduces works also under the consideration of disappearing edges and the model adaptions we introduced.

This is due to the fact that if there is at most one cycle, a disappearing edge can only break the cycle or split a connected component in two connected components. Therefore each connected component $C(G_a)$ of the altered graph G_a can be seen as an unaltered graph and verified individually, if it contains a cycle or not.

THEOREM 9. *The proof size for 1-*CYCLIC *is 2 bit for any number $k \leq |E_i|$ of edges disappearing. $|E_i|$ denotes the number of edges in the initial graph.*

[8] proves that the proof size for CYCLIC in undirected graphs is at least 2 bits. Since their model does not consider any inconsistencies and is therefore stronger, this lower bound holds also in our model. To prove Theorem 9 we are left to prove the following lemma:

LEMMA 10. *There exists a correct PVP with a proof size of 2 bit that is correct for 1-*CYCLIC *if any number $k \leq |E_i|$ of edges might disappear. $|E_i|$ denotes the number of edges in the initial graph.*

Due to space constraints, we defer the proof to the full version of this article. See Figure 9 for an example of the proof labeling used.

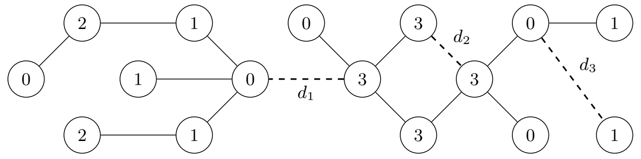
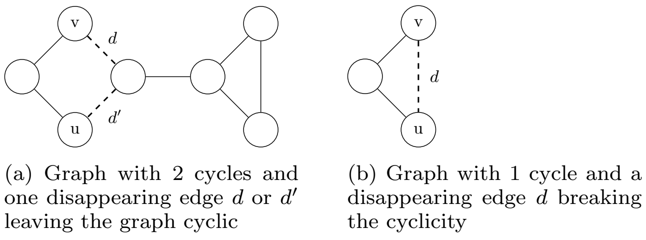


Figure 9: Example graph showing the 2 bit proof labeling introduced by [8] to verify cyclicity. We take an analogous PVP showing that it is correct for 1-CYCLIC considering that edges might disappear, e.g., d_1, d_2 and/or d_3. Here, vertices on the cycle are labeled 3 and vertices not on the cycle are labeled with their distance modulo 3 to the root of their subtree, where the root is a vertex next to the cycle.



(a) Graph with 2 cycles and one disappearing edge d or d' leaving the graph cyclic

(b) Graph with 1 cycle and a disappearing edge d breaking the cyclicity

Figure 10: Example graphs to show why no PVP can be correct for CYCLIC if one edge might disappear. A prover would have to give v in Figure 10a a label for which the verifier outputs YES with adjacent disappearing edge d since there is a connection through u to the other cycle. The same holds for the label u gets since if d' disappears there is a connection through v to the other cycle. Such a labeling would however output YES on all vertices in Figure 10b.

4.1.2 Multiple Cycles

The problem gets much more complicated when more than one cycle can exist in the graph. For this we consider now the case where the initial graph $G_i \in$ CYCLIC can have any amount of cycles and at most one edge d might disappear.

We take as example a connected initial graph G_i that has two cycles and note that the graph is still cyclic even if one of the cycles breaks due to the disappearing edge. Therefore, vertices on each cycle must get some information from the labeling that there exists another cycle in the graph. This however leads to an insolvable problem in our model as we will show. More precisely:

THEOREM 11. *There exists no correct PVP for* CYCLIC *as specified in our model that can consider one disappearing edge.*

PROOF. Assume for the sake of contradiction that there exists a correct PVP $(\mathcal{P}, \mathcal{V})$ as required. Consider the example graphs shown in Figure 10:

Given the initial graph $G_{i,(a)}$ in Figure 10a the prover $\mathcal{P}$ would have to give the vertex v a label $l(v)$, for which the verifier $\mathcal{V}$ computes YES if it senses an adjacent disappearing edge d, i.e., if the first inconsistency reporting bit $r_1(v)$ is set. Such a label $l(v)$ is needed since if d disappears in Figure 10a there is still a path from v through u to the other cycle in the graph. (Note that at most one edge disappears, so either d or d'). However, a similar label $l(u)$ is needed for the vertex u since d' might disappear instead of d and u would than still have a path through v to the other cycle. Therefore v and u must have labels for which $\mathcal{V}$ computes YES if there is an adjacent disappearing edge. This however would be a labeling for which $\mathcal{V}$ computes YES on every vertex in Figure 10b. This contradicts that $\mathcal{V}$ should compute NO for at least one vertex for any connected component $C(G_a) \notin$ CYCLIC, regardless of the labeling. □

Given that there is no correct PVP for CYCLIC that can consider one disappearing edge, the question about a correct PVP for CYCLIC that can consider multiple disappearing edges becomes redundant.

4.2 One Appearing Edge

In this subsection we take a look at how cyclicity might change if at most one edge a appears, i.e., $E(G_a) = E(G_i) \vee E(G_i) \cup \{a\}$ where a can appear between any two vertices $u, v \in V$. Before we prove or disprove a PVP, note the following facts:

- A connected component containing a cycle cannot become acyclic through an appearing edge
- A connected component $C_1(G_i) = (V_{C1}, E_{C1})$ containing no cycle can only get cyclic if it has an appearing edge adjacent to one of its vertices and either the appearing edge connects $C_1(G_i)$ to another connected component $C_2(G_i)$ containing a cycle or both end vertices u and v of the appearing edge are within the connected component, i.e. $u, v \in V_{C1}$.
- if both end vertices u and v of the appearing edge are within the connected component, $C_1(G_a)$ is for sure cyclic. This holds since every acyclic connected component $C = (V, E)$ has to have exactly the minimal number of edges $|E| = |V| - 1$ to be connected and acyclic. Any additional edge closes a cycle.

Given these facts we note that a vertex v on one side of the appearing edge must have the ability to distinguish between initially separated connected component to know, whether u on the other side of the appearing edge belongs to the same connected component or not.

Since this cannot be achieved through a potentially random labeling we allow for *IDs* in this subsection. Thereby an *ID* is a unique identifier every vertex $v \in V(G_i)$ of the initial graph gets attached to itself, where the uniqueness is guaranteed by the model. Further, we adjust our model such that the set of received messages $M(v)$ on vertex v not only includes labels, but also the IDs of each neighbor of v.

By adding unique identifiers as described to the model, we can now show the existence of a PVP. The technical proof is deferred to the full version of this article.

THEOREM 12. *There exists a PVP, using unique identifiers, with a proof size in $\mathcal{O}(n^2 \log n)$ bit that is correct for* CYCLIC *if at most one edge a appears.*

4.3 Multiple Inconsistencies

In this subsection we focus on PVPs for cyclicity considering more than one inconsistency. We already looked at multiple disappearing edges in Section 4.1. Therefore, it is left to discuss multiple appearing edges and combinations of disappearing and appearing edges.

4.3.1 *Multiple Appearing Edges*

We consider the initial graph G_i that is a set of connected components and ask whether there is a correct PVP that

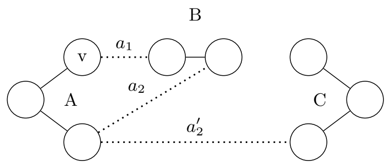


Figure 11: Graph showing that cyclicity cannot be verified in a local manner if there might be two edges appearing. A, B and C denote the three acyclic connected components of the initial graph. Here, vertex v can sense an adjacent appearing edge a_1, but cannot receive enough information within one round of communication to decide if a second appearing edge a_2 exists that would lead to cyclicity. Even if v would get the information of another appearing edge, it could also be that the second edge that appeared is a_2' (and not a_2), leaving the resulting connected component acyclic.

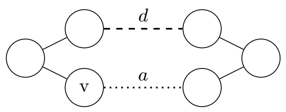


Figure 12: Graph showing that cyclicity cannot be verified in a local manner if there is an edge appearing and there might be an edge disappearing. v can sense an adjacent appearing edge a, but cannot receive the information to decide if a disappearing edge d on the other side of the graph exists that would leave the graph acyclic.

considers multiple appearing edges and verifies for each connected component $C(G_a)$ of the altered graph G_a individually, if $C(G_a) \in$ CYCLIC. This, having the set of edges in the altered graph $E(G_a) = E(G_i) \cup A$ where A denotes the set of appearing edges. Remember that appearing edges can appear anywhere in the graph and therefore cannot be located within the graph by a non-adjacent vertex. We refer to Figure 11 for an illustration of the deferred proof.

THEOREM 13. *There exists no correct PVP for* CYCLIC *as specified in our model that can consider more than one appearing edge.*

4.3.2 *Combination of Dis– and Appearing Edges*

Since there is no correct PVP considering disappearing edges if the graph contains more than one cycle (see Section 4.1.2) and appearing edges tend to form cycles, the question about a correct PVP for cyclicity that considers a combination of dis– and appearing edges is rather complex.

However if we focus on at most one edge disappearing and at most one edge appearing, we can already show that no correct PVP for 1-CYCLIC can be found. This due to the fact that if there is an appearing edge a forming a cycle, vertices adjacent to a cannot know if there is a disappearing edge on the other end of the graph breaking the cycle again. We refer to Figure 12 for an illustration of the deferred proof.

THEOREM 14. *There exists no correct PVP for 1-*CYCLIC *as specified in our model that can consider more than one inconsistency of any type if at least one edge is appearing.*

5. RELATED WORK

The concept of local checkability has a rich history, starting with the seminal paper of Naor and Stockmeyer [18]:

They coined the term *Locally Checkable Labelings (LCL)*, checking labels locally in a constant number of rounds.

Further models for local checkability have been introduced since then, extending and building upon the ideas of [18]. We now cover the four ones closest related to our work:

Göös and Suomela [14, 15] allowed $f(n)$ bits of additional information on the vertices introducing *Locally Checkable Proofs (LCP)* as an addition to Locally Checkable Labelings ($f(n) = 0$ being LCL). They investigated on distributed local decision problems: which proof size $f(n)$ is needed such that all vertices output Yes on a Yes-instance and at least one vertex outputs No on any invalid proof or No-instance.

With *Proof Labeling Schemes* Korman et al. [16] restricted the LCP model to allow for one communication round only. Thus, the verification of the proof labels are computed from each vertex' immediate neighborhood. Super-constant communication rounds have also been studied in [3].

Both the models of [14] and [16] also investigate the use of unique identifiers, however a distinction to our work is that they always assume a port numbering to be given. Further studies on more universal identifiers were performed in [12]. Cf. also [7] for a generalization, coined local hierarchy *LH*.

Fraigniaud et al. [11] took a different approach by allowing multiple communication rounds but separating the proof labels from vertex identifiers. They distributed the labels by having only the graph structure without identifiers given, but the verifier is aware of the identifiers. Unlike Göös and Suomela [14], and Korman et al. [16], Fraigniaud et al. studied the impact of randomization on local checkability as well, also regarding reduced proof sizes [2, 10, 13].

Most related to this work is the article of Foerster et al. [8, 9], which uses a combination of the above as model. By having a *Prover P* distributing proof-labels and a *Verifier V* checking the labels after one communication round, they introduce the *Prover-Verifier-Pairs (PVPs)* we adapted for this work. These have *no strings attached* meaning that they rely on neither identifiers nor port numbers of any sort to make decision. Unlike [8], our work focuses on undirected graphs, but introduces inconsistencies to the network graph.

For a recent and encompassing overview on distributed decisions beyond the works discussed above we refer to the survey of Feuilloley and Fraigniaud [6]. A further field related to local checkability is property testing, cf. [5].

We are not aware of work that connects local checkability with dynamic networks. Dynamic networks are in the focus of research for a long time [1], especially since the rise of peer-to-peer computing, but their study in the local model [4, 17] is more recent.

6. CONCLUDING REMARKS

We discussed the applicability of local checkability to dynamic networks, where edges might appear or disappear when verifying network properties. To the best of our knowledge, our work is the first that extends local checking to dynamic network changes. As thus, we focused on (im-)possibility results – showing that there are cases where quite simple schemes suffice, but that problems which seem equally "easy" at first glance can also allow for no local solution at all, with the occasional corrective supplied by unique identifiers beyond the possibilities of a simple prover-verifier-pair. We note that the restriction to one communication round does not fundamentally change the theoretical power of a PVP in our case: The provided counter-examples can be extended by replacing edges with paths of nodes of length k, for any constant k, s.t. problems in NLC stay in NLC. We believe local checkability in dynamic networks to be an intriguing field of study, as it extends the fascinating theory of local verification towards the realm of topology changes.

Acknowledgments

We would like to thank the anonymous reviewers of ICDCN 2017 for their helpful comments on our submitted manuscript.

7. REFERENCES

[1] B. Awerbuch, Y. Mansour, and N. Shavit. Polynomial end-to-end communication. In *FOCS*, 1989.

[2] M. Baruch, P. Fraigniaud, and B. Patt-Shamir. Randomized proof-labeling schemes. In *PODC*, 2015.

[3] M. Baruch, R. Ostrovsky, and W. Rosenbaum. Brief announcement: Space-time tradeoffs for distributed verification. In *PODC*, 2016.

[4] R. O. Bischoff and R. Wattenhofer. Information Dissemination in Highly Dynamic Graphs. In *DIALM-POMC*, 2005.

[5] K. Censor-Hillel, E. Fischer, G. Schwartzman, and Y. Vasudev. Fast distributed algorithms for testing graph properties. *CoRR*, abs/1602.03718, 2016.

[6] L. Feuilloley and P. Fraigniaud. Survey of distributed decision. *CoRR*, abs/1606.04434, 2016.

[7] L. Feuilloley, P. Fraigniaud, and J. Hirvonen. A hierarchy of local decision. *ICALP*, 2016.

[8] K.-T. Foerster, T. Luedi, J. Seidel, and R. Wattenhofer. Local Checkability, No Strings Attached. In *ICDCN*, 2016.

[9] K.-T. Foerster, T. Luedi, J. Seidel, and R. Wattenhofer. Local Checkability, No Strings Attached: (A)cyclicity, Reachability, Loop Free Updates in SDNs. *Theor. Comput. Sci.*, To appear.

[10] P. Fraigniaud, M. Göös, A. Korman, M. Parter, and D. Peleg. Randomized distributed decision. *Distributed Computing*, 27(6):419–434, 2014.

[11] P. Fraigniaud, M. Göös, A. Korman, and J. Suomela. What can be decided locally without identifiers? In *PODC*, 2013.

[12] P. Fraigniaud, J. Hirvonen, and J. Suomela. Node labels in local decision. In *SIROCCO*, 2015.

[13] P. Fraigniaud, A. Korman, and D. Peleg. Towards a complexity theory for local distributed computing. *J. ACM*, 60c(5):35, 2013.

[14] M. Göös and J. Suomela. Locally checkable proofs. In *PODC*, 2011.

[15] M. Göös and J. Suomela. Locally checkable proofs. *Theory of Computing*, To appear.

[16] A. Korman, S. Kutten, and D. Peleg. Proof labeling schemes. *Distributed Computing*, 22(4):215–233, 2010.

[17] F. Kuhn, N. Lynch, and R. Oshman. Distributed computation in dynamic networks. In *STOC*, 2010.

[18] M. Naor and L. J. Stockmeyer. What can be computed locally? In *STOC*, 1993.

[19] S. Schmid and J. Suomela. Exploiting locality in distributed SDN control. In *HotSDN*, 2013.

[20] N. Shelly, B. Tschaen, K.-T. Foerster, M. A. Chang, T. Benson, and L. Vanbever. Destroying networks for fun (and profit). In *HotNets*, 2015.