Local Checkability, No Strings Attached:[☆] (A)cyclicity, Reachability, Loop Free Updates in SDNs

Klaus-Tycho Foerster[a,b], Thomas Luedi[a], Jochen Seidel[a], Roger Wattenhofer[a]

[a] *ETH Zürich, 8092 Zurich, Switzerland*
[b] *Microsoft Research, 98052 Redmond, WA, USA*

Abstract

In this work we study local checkability of network properties like s-t reachability, or whether the network is acyclic or contains a cycle. A structural property S of a graph G is *locally checkable*, if there is a prover-and-verifier pair $(\mathcal{P}, \mathcal{V})$ as follows. The *prover* $\mathcal{P}$ assigns a label to each node in graphs satisfying S. The *verifier* $\mathcal{V}$ is a constant time distributed algorithm that returns Yes at all nodes if G satisfies S and was labeled by $\mathcal{P}$, and No for at least one node if G does not satisfy S, regardless of the node labels. The quality of $(\mathcal{P}, \mathcal{V})$ is measured in terms of the label size.

Our model has *no strings* attached, i.e., we do not assume any identifiers or port numbers: All we allow is a single exchange of labels between neighbors.

We obtain (asymptotically) tight bounds for the bit complexity of the latter two problems for undirected as well as directed networks, where in the directed case we consider one-way and two-way communication, i.e., we distinguish whether communication is possible only in the edge direction or not. For the one-way case we obtain a new asymptotically tight lower bound for the bit complexity of s-t reachability, which also extends to distributed algorithms with constant time execution. For the two-way case we devise an emulation technique that allows us to transfer a previously known s-t reachability upper bound without asymptotic loss in the bit complexity.

Lastly, we also study how to apply the concept of local checkability to updating spanning trees in a loop free manner in the context of asynchronous networking, by exploring the similarities between prover-and-verifier pairs and Software Defined Networks (SDNs).

Keywords:
Local Checking, s-t Reachability, Acyclicity, Software Defined Networking

---

[☆] A preliminary extended abstract appeared in the Proceedings of the 17th International Conference on Distributed Computing and Networking (ICDCN '16), ACM, New York, NY, USA, `http://dx.doi.org/10.1145/2833312.2833315` [18].

Email addresses: `foklaus@ethz.ch` (Klaus-Tycho Foerster), `tluedi@ethz.ch` (Thomas Luedi), `seidelj@ethz.ch` (Jochen Seidel), `wattenhofer@ethz.ch` (Roger Wattenhofer)

1. Introduction

Network administrators must know whether the network is correct [45], e.g. whether destination t is reachable from source s, or whether the forwarding rules present in the network imply that packets may potentially be sent in a cycle. Often such network properties are checked by constantly sending probe packets into the network, or, alternatively, by sending the state of all nodes in the network to a central location where all the data is then verified. Both methods take time, often too much time. It would be advantageous to perform these costly global operations only if needed – and otherwise rely on inexpensive local verification [44]. Our paper studies local checkability of fundamental structural properties for directed as well as undirected networks: Nodes of a network can check whether a given *global* structural property of a network is guaranteed, just by *locally* comparing their state with the state of their neighbors.

The concept of local checkability was popularized by Naor and Stockmeyer in [41]. In our context, this concept refers to the nodes' ability to decide (verify) whether the network has the desired property by exchanging *labels* with their neighbors. The nodes decide YES if all nodes agree, and NO if at least one node disagrees. In practice the disagreement could subsequently be reported. With deterministic algorithms only few properties can be checked locally. If however nodes are allowed to use (a bounded amount of) nondeterminism, a rich complexity hierarchy arises [29]. We focus on the fastest possible case where nodes are only allowed to communicate a single round, cf. [35]. Furthermore, our model has *no strings attached*, i.e., we do not assume any identifiers or port numbers: All we allow is a single exchange of labels between neighbors.

To obtain a better understanding of nondeterminism in the context of distributed computing, let us quickly explain a toy example. Consider the set BIPARTITE containing all bipartite graphs. In the sequential setting, BIPARTITE would be called a *language*, and the YES-instances (*words*) in BIPARTITE are exactly the graphs that allow a bipartition of the nodes. As in the sequential setting, one may now ask: Is there a (nondeterministic) distributed algorithm deciding whether a given graph G is in BIPARTITE, using only a single communication round? Indeed, such an algorithm exists [29]. First each node v nondeterministically chooses either the value 0 or 1 and sends it to all neighbors. Next, v checks if all its neighbors sent the value *not* chosen by v.

The proposed nondeterministic algorithm indeed decides BIPARTITE. A bipartition of the graph corresponds to a nondeterministic choice of 0 and 1 for every node v so that all neighbors of v choose the opposite value. Thus, when the graph G is bipartite, the nodes nondeterministically decide YES. On the other hand, if G is not bipartite, then in all possible nondeterministic choices of the nodes, at least two nodes will have a neighbor that chose the same value. In that case, the nodes decide NO.

Every nondeterministic distributed algorithm can be expressed as a deterministic algorithm with access to a *proof labeling* [29], where the proof labeling corresponds to an oracle in the sequential setting. More precisely, a nondeterministic algorithm is a pair $(\mathcal{P}, \mathcal{V})$, referred to as *prover-verifier pair (PVP)*.

Decision Problem	Directed one-way	Directed two-way	Undirected	
s-t reachability	$\Theta(\log n)$	$O(\log \Delta)$	1 [29]	Section 3
Contains a Cycle	not possible	$\Theta(\log n)$	2	Section 2.1
Acyclic	$\Theta(\log n)$	$\Theta(\log n)$	same as Tree	Section 2.2
Tree	$\Theta(\log n)$ [35]	$\Theta(\log n)$	$\Theta(\log n)$ [35]	Section 2.3

Table 1: The proof label size (in bits) necessary and sufficient for a PVP with respect to different graph decision problems and communication primitives. Here n denotes the number of nodes in the network G, and Δ is the maximum degree of any node in G. For s-t reachability, the $O(\log \Delta)$ one-way upper bound with port numbers [29] translates to our two-way model, see Section 3. For Trees, the $O(\log n)$ upper bound for directed one-way communication from [35] also applies in the two-way model.

The task of the *prover* $\mathcal{P}$ is to assign labels to nodes (the *proof*) in a YES-instance. The *verifier* $\mathcal{V}$ gets as input at node v only the labels of v and its neighbors. Now $\mathcal{V}$ has to decide YES (at all nodes) in YES-instances labeled by $\mathcal{P}$; In NO-instances $\mathcal{V}$ has to decide NO (for at least one node) regardless of the node labels.

The complexity of such nondeterministic algorithms is measured in terms of the maximum *proof label size* used by $\mathcal{P}$. This corresponds to the number of nondeterministic choices made throughout the execution. In our BIPARTITE example each node only needs a single bit[1] as its label.

There are two ways to view communication in directed graphs: Nodes can communicate only in the direction of the edge (*directed one-way* communication), or the edge direction imposes no restrictions for communication but only for the network property itself (*directed two-way* communication). We investigate both cases, as well as the undirected case, where nodes communicate with all their neighbors. One of our findings is that all three models are fundamentally different, not only in terms of proof label size, but also in terms of decidability. The results for each of our three network structure detection problems are summarized in Table 1.

Another result of our work is the first non-trivial asymptotically tight lower bound for the directed s-t reachability [2] problem that does not rely on descriptive complexity methods. In that problem, two nodes s and t are guaranteed by the problem setting, and the question is whether there is a directed path from s to t. Note that both the directed and the undirected variant are well understood in terms of descriptive complexity, and the directed variant is known to be more difficult [2, 10]. While the observations from [10] lead to a proof label size of 1-bit for the undirected variant, showing a non-trivial lower bound for the directed case remained an open question. Our bounds can be extended to distributed algorithms beyond the restriction to a single round of communication: We show in Section 4 that when allowing k rounds of communication, with k being any constant, the bounds on the label size remain at $\Theta(\log n)$.

[1] Note that a standard covering argument (the 6-cycle is bipartite, while the 3-cycle is not) can be used to show that one nondeterministic choice is also necessary.

In light of our tight $\Theta(\log n)$ bound for the s-t reachability problem with directed one-way communication we revisit the $O(\log \Delta)$ bound from [29]. In particular, their upper bound relies on the fact that the underlying communication mechanism discloses port numbers to the verifier. As we will detail in Section 3, this is not necessary: When directed two-way communication is available, the label can be extended to include *checkable* port numbers using only $O(\log \Delta)$ additional bits. Since referring to a single port number requires $\log \Delta$ bits anyway, this does not change the asymptotic label size.

Lastly, in Section 5, we extend a line of work started by Schmid and Suomela in [44]: As it turns out, networks managed by a central controller, also known as Software Defined Networks (SDNs), show a strong similarity to prover-verifier pairs. The controller can take on the role of the prover, and the switches in the network itself the role of the verifiers. We show how the concept of local checkability can be used for graceful network reconfigurations with the example of migrating in a loop[2] free manner between forwarding rules.

1.1. Related Work and Background on Local Checkability

More than 20 years ago, Naor and Stockmeyer [41] raised the question of "*What can be computed locally*?" In their work, the notion of *Locally Checkable Labelings (LCL)* is investigated, where labels are checked in a local fashion, i.e., in a constant number of communication rounds.

This line of research is being followed in many directions, with the concepts of *Proof Labeling Schemes (PLS)*, *Nondeterministic Local Decisions (NLD)*, and *Locally Checkable Proofs (LCP)* being most related to our work. We note that all three approaches are strictly stronger than the model discussed in this paper (by adding either identities, port numbers, or more potent communication models).

The term Locally Checkable Proofs was coined by Göös and Suomela [29] as an extension to Locally Checkable Labelings, where $\mathrm{LCP}(f)$ allows for $f(n)$ bits of additional information per node. They study decision problems from the viewpoint of nondeterministic distributed local algorithms: Is there a proof of size $f(n)$ such that all nodes will output Yes for Yes-instances, with any (invalid) proof for a No-instance being rejected by at least one node? The authors introduce a complexity hierarchy for various problems, with $\mathrm{LCP}(0)$ being equivalent to LCL. For most of the results in [29], unique identifiers are assumed for each node, or at least port numbers – which can be used for verification purposes. Thus, their algorithms may use additional strings of information free of cost, which might not be relevant asymptotically for large proof sizes, but come into play for small labels: E.g., in the case of directed s-t reachability, they show that $O(\log \Delta)$ bits suffice by "pointing" at the successor node in the s-t path, a technique relying on port numbers.

The Proof Labeling Schemes of Korman et al. [34, 35] differ from LCPs in the sense that they only use one round of communication to transfer the labels.

[2]In the context of SDNs, the term loops for cycles is prevalent, which is why we will use the notion of loops when talking about cycles in the forwarding rules of SDNs.

Thus, upper bounds from PLS apply to LCP and lower bounds from LCP apply to PLS, as the LCP model is strictly more powerful than the PLS model. In [35], the authors also investigate the role of unique identities in PLS and show that there are cases where (given) unique identities are necessary, but also examples where the transition to identities is possible. Another difference between LCP and PLS is that in PLS, the identities of the neighboring nodes of a node v are not available to the verifier at v. Nonetheless, they assume the nodes to be aware of the port numbers of their edges.

Closely related to our work, they study (among other problems) the question of whether a connected subgraph is a tree and give asymptotically matching upper and lower bounds of $\Theta(\log n)$ bits for directed one-way communication and the undirected case. Their proofs and techniques for trees carry over to the model considered in this paper and are thus referenced in Table 1. For spanning tree verification (also without the notion of labels, cf. [43]), the construction in [35] is also used in the context of Software Defined Networks (SDNs) [44]: Inconsistencies of a spanning tree for routing can be detected locally, triggering a (costly) global recomputation only if needed.

Nondeterministic Local Decisions [26] considers distributed nondeterminism for decision problems. Like LCP and unlike PLS, they allow more than one communication round. However, the proofs are not allowed to depend on the identifier of a node (see [23, 24] for the impact of (missing) identifiers on local decisions). Like in our case the nodes are anonymous to the prover, but unlike in our case they are not anonymous to the verifier. In some sense, as described by [29], the class NLD for connected graphs can be understood as LCL $\subsetneq$ NLD $\subsetneq$ LCP(∞). Unlike LCP and PLS above, Fraigniaud et al. [26] also study the impact of randomization. Among many other results, they reveal surprising connections between randomization and oracles related to nondeterministic computing: As it turns out, an oracle providing the nodes with the size of the graph gives *"roughly* [. . .] *the same power to nondeterministic distributed computing as randomization does"* [26]. Additional recent results concerning the power of randomization for local distributed computing can be found in [9, 22]. A generalization of identifiers, so-called scalar oracles, were studied in [25].

Furthermore, there exists a strong connection between proof labeling schemes and self-stabilization (we refer to [16] for an overview of the topic): As characterized by Blin et al. [11], *"any mechanism insuring silent self-stabilization is essentially equivalent to a proof-labeling scheme"*. Even more so, the proof size nearly corresponds to the number of registers for self-stabilization [11]. As such, there has been a long line of research connecting local checking with self-stabilization [1, 6, 7, 8].

We ask the question of how a global prover can convince a distributed verifier that it fulfills a certain property. One may also ask the converse question, i.e., how a distributed prover could convince a centralized verifier that knows only node labels, but not the graph structure. This inverted setting is studied in the works of Arfaoui et al. for trees [5] and cycle-freeness [4].

1.2. Preliminaries

Graphs and Node Labels. We model the network as a graph $G = (V(G), E(G))$, where $V(G)$ and $E(G)$ denote the set of vertices and edges, respectively, and when G is clear from the context, we write $V = V(G)$ and $E = E(G)$. Similarly, we write $n = n(G) = |V(G)|$ for the number of nodes in G. The graph G may be either directed or undirected, but we always assume G to be (weakly[3]) connected. For a node $v \in V$, we denote by $\deg_{\text{in}}(v)$ and $\deg_{\text{out}}(v)$ the number of incoming and outgoing edges of v in G, respectively. We set $\deg(v) = \deg_{\text{in}}(v) = \deg_{\text{out}}(v)$ if G is undirected, and $\deg(v) = \deg_{\text{in}}(v) + \deg_{\text{out}}(v)$ if G is directed. By $\Delta(G) = \max_{u \in V} \deg(u)$ (or simply Δ) we denote the maximum degree in G.

For two nodes $u, v \in V$, let $\text{dist}(u, v)$ denote the distance between both nodes in G (regarding the distance function in the underlying undirected graph in the directed case). A *(node) labeling* for G is a function $\ell : V \to \{0, 1\}^*$ that assigns a finite *label* (i.e., a bit-string) to every node in V.

Communication Means. Let G be a graph, let ℓ be a node labeling for G, and let v be a node in G. We now consider three means of communication in G, namely U, D_1, and D_2, corresponding to *undirected*, *one-way*, and *two-way* communication, respectively. If G is undirected, then $U(v)$ is the multiset $[\ell(u_1), \ldots, \ell(u_k)]$ containing $\deg(v)$ labels, where $u_1, \ldots, u_k$ are the neighbors of v. If G is directed, then we distinguish two cases. For directed one-way communication, $D_1(v)$ is the multiset $[\ell(u_1), \ldots, \ell(u_k)]$ containing $\deg_{\text{in}}(v)$ labels, where $u_1, \ldots, u_k$ are the in-neighbors of v. For directed two-way communication, $D_2(v)$ is a pair (I, O), where I is $D_1(v)$ and O is the multiset containing $\deg_{\text{out}}(v)$ many labels of v's out-neighbors. I.e., the sets $U(v), D_1(v)$, and $D_2(v)$ are the *messages* received by v when the corresponding communication method is used. We denote the empty multiset by $[\,]$.

Observe that all multisets above are unordered, i.e., there are no unique identifiers and there is no notion of port labels on the edges. If such an order is necessary (for some verifier), then the means to order the multiset need to be included in the proof labels, since the communication mechanism itself does not attach any strings to the messages. In the directed two-way case, however, there is a clear distinction between messages transferred along the edge direction or opposite to it. Note that this distinction is necessary: If it was not made, the directed two-way mode would essentially be equivalent to the undirected case, since the edge direction becomes indistinguishable.

Local Checkability. An *(un)directed network property* is specified by a set Y of (un)directed graphs containing the Yes-*instances*, and any (un)directed graph $G \notin Y$ is referred to as a No-*instance*. A *prover-verifier pair* $(\mathcal{P}, \mathcal{V})$ *for* Y (*PVP* for short) works as follows.

The *prover* $\mathcal{P}$ gets as an input a graph $G \in Y$ and computes a (finite) node label $\ell(v)$ for every $v \in V$. This labeling ℓ obtained from $\mathcal{P}$ is referred to as

[3]A directed graph is called weakly connected if the underlying undirected graph is connected.

proof. Let G be any graph, and let ℓ be any node labeling for G. The *verifier* $\mathcal{V}$ is a distributed algorithm that gets as an input at node v the label $\ell(v)$; and in addition either $U(v)$ if Y is an undirected property, or $D_1(v)$ respectively $D_2(v)$ depending on the communication means if Y is a directed property.

A PVP $(\mathcal{P}, \mathcal{V})$ is *correct for* Y if it satisfies

(1) if $G \in Y$ and ℓ was obtained from $\mathcal{P}$, then $\mathcal{V}$ returns YES at all nodes; and

(2) if $G \notin Y$, then $\mathcal{V}$ returns NO for at least one node, regardless of the node labels.

Whenever necessary, we specify the PVP by the communication means used for the verifier, and write *U-PVP*, *D_1-PVP*, and *D_2-PVP* correspondingly. When $X \in \{U, D_1, D_2\}$ is some means of communication, then a network property Y is *X-locally checkable* if there is a correct X-PVP for Y.

The quality of a PVP is measured in terms of the maximum label size in bits assigned by the prover. For a PVP $(\mathcal{P}, \mathcal{V})$, the *proof size of* $(\mathcal{P}, \mathcal{V})$ is $f(n)$ if the labels assigned by $\mathcal{P}$ use at most $f(n)$ bits in any YES-instance containing at most n nodes. For a network property Y, the *X-proof size for Y* is the smallest proof size for which there exists a correct X-PVP for Y. Since the communication means are clear for undirected properties, we omit them in that case. Throughout this paper, all logarithms use base 2 and are rounded up to be of integer value.

2. Checking Network Properties

2.1. Cycles

Let U-CYCLE denote the set of all undirected connected graphs containing at least one cycle. Let correspondingly D-CYCLE denote the set of all weakly connected directed graphs containing at least one directed cycle. Note that an undirected graph is in U-CYCLE exactly if it is not an undirected tree, while a directed graph G is in D-CYCLE exactly if G is not a directed acyclic graph (DAG). In the remainder of this section we establish the following:

Theorem 1. *For the cycle detection problem, it holds that*

(i) There is no D_1-PVP for D-CYCLE.

(ii) The D_2-proof size for D-CYCLE *is* $\Theta(\log n)$ *bits.*

(iii) The U-proof size for U-CYCLE *is* 2 *bits.*

We prove each claim listed in Theorem 1 separately, starting with the directed cases. As the first step we show that there cannot be a D_1-PVP for D-CYCLE.

Lemma 2. *There is no D_1-PVP for* D-CYCLE.

Proof. Assume, for the sake of contradiction, that there exists a correct D_1-PVP $(\mathcal{P}, \mathcal{V})$ for D-CYCLE. Our goal is to construct a NO-instance H and node labels ℓ' for the nodes in H so that $\mathcal{V}$ returns YES at all nodes. To that end, consider the YES-instance G (depicted in Figure 1) consisting of a cycle with two nodes

c_1, c_2, and two additional nodes a, b, where a has the two outgoing edges (a, c_1) and (a, b). Let ℓ denote the node labeling assigned to G by $\mathcal{P}$, and denote by A and B the values $\ell(a)$ and $\ell(b)$, respectively.

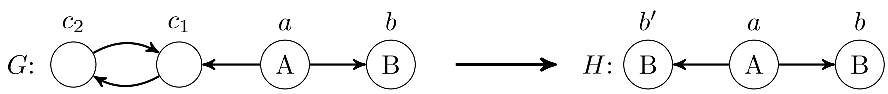

Figure 1: Yes-instance G and No-instance H of D-Cycle. A and B are the labels assigned to the nodes a and b in G by the prover $\mathcal{P}$, the node labels in the cycle are not shown. The construction of H yields that for each u in H there is a v in G with $(\ell'(u), D_1(u)) = (\ell(v), D_1(v))$.

Our No-instance H, as shown in Figure 1, consists of the three nodes a, b, and b', and the two edges (a, b) and (a, b'). Note that indeed, H does not contain a cycle. By assigning the labels $\ell'(a) = \text{A}$ and $\ell'(b) = \ell'(b') = \text{B}$, we obtain that for all nodes u in H there is a corresponding node v in G for which $(\ell'(u), D_1(u)) = (\ell(v), D_1(v))$. The verifier $\mathcal{V}$ can therefore not differentiate between u and v and thus returns Yes for all nodes in H. This contradicts the assumption that $(\mathcal{P}, \mathcal{V})$ is correct for D-Cycle. □

Lemma 3. *There is a D_2-PVP for* D-Cycle *with a proof size of* $\log n$ *bits.*

Proof. We describe a D_2-prover-verifier pair $(\mathcal{P}, \mathcal{V})$ for D-Cycle as required. Let $G = (V, E) \in$ D-Cycle and let $C \subseteq V$ be the set of all nodes that are in a directed cycle. The prover $\mathcal{P}$ labels all nodes $v \in V$ as follows. First, all nodes $v_c \in C$ are labeled with $\ell(v_c) = 0$. All other nodes $v \in V$ are labeled regarding their distance to the closest cycle: The prover $\mathcal{P}$ sets $\ell(v) = \text{dist}_C(v)$, where $\text{dist}_C(v) = \min_{v_c \in C} \text{dist}(v_c, v)$. We refer to Figure 2 for an example. As the distance is bounded from above by n, the maximum label size is $\log n$ bits.

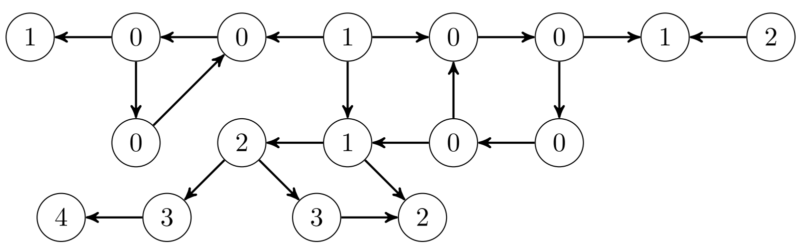

Figure 2: A Yes-instance of D-Cycle labeled for two-way communication. All nodes on cycles have the label 0, and all other nodes are labeled with the minimum distance to the nearest cycle using the distance function in the underlying undirected graph.

The verifier $\mathcal{V}$ returns Yes for nodes v_c with $\ell(v_c) = 0$ if for the received pair (I, O) of labels holds: There is a label of 0 in I and a label of 0 in O. For the other nodes $v \in V$, Yes is returned by $\mathcal{V}$ if a) there is an edge (u, v) or (v, u) such that $\ell(v) = \ell(u) + 1$ and b) no edge (u', v) or (v, u') such that $\ell(v) > \ell(u') + 1$. In all other cases, $\mathcal{V}$ returns No.

We now show that $\mathcal{V}$ returns YES for all nodes v in YES-instances that were labeled by the prover $\mathcal{P}$: The prover $\mathcal{P}$ labeled only (and all the) nodes on a directed cycle with a 0, i.e., if $\ell(v) = 0$, then $\mathcal{V}$ returns YES for v. The remaining case is $\ell(v) = j > 0$. If $\ell(v) = j$, then $\mathrm{dist}_C(v) = j$, i.e., there exists a node $u \in V$ such that $\mathrm{dist}_C(v) = \mathrm{dist}_C(u) + 1$ and no node $u' \in V$ such that $\mathrm{dist}_C(v) > \mathrm{dist}_C(u') + 1$, as by the definition of $\mathcal{P}$. Thus, $\mathcal{V}$ returns YES as well.

For the D_2-PVP $(\mathcal{P}, \mathcal{V})$ to be correct, it is left to show that $\mathcal{V}$ returns NO for at least one node if the considered graph is not in D-CYCLE. Analogously to the undirected case, let G_{no} be a weakly connected directed graph containing no directed cycle.

For contradiction, assume there would be a node $v \in V(G_{no})$ with $\ell(v) = 0$. Then there has to be a node v_1 with $\ell(v)_1 = 0$ such that there exists an edge (v, v_1), else $\mathcal{V}$ would return NO. This concept of "following the zero" can be iterated, but as the graph is finite (and does not contain a directed cycle), there will be a node v_j for which no node v_{j+1} with $\ell(v_{j+1}) = 0$ exists such that there is an edge (v_j, v_{j+1}). Hence, $\mathcal{V}$ would return NO and therefore no node can be labeled with 0 in G_{no}.

An idea similar to following the zero can now be applied again: W.l.o.g., let v be a node with the label k. There has to be an edge (v, v_1) with $\ell(v_1) = k - 1$, else $\mathcal{V}$ would return NO for v. Again, as the graph is finite and contains no cycle, following the outgoing edge to a decreasing label is no longer possible at some point. Thus $\mathcal{V}$ will return NO for any weakly connected directed graph not containing a cycle, meaning that the D_2-PVP $(\mathcal{P}, \mathcal{V})$ is correct. □

Lemma 4. *The D_2-PVP proof size for* D-CYCLE *is at least* $\log\left(\frac{n-5}{2}\right)/2$ *bits.*

We establish Lemma 4 by showing that any D_2-PVP $(\mathcal{P}, \mathcal{V})$ with a smaller proof size can be fooled. To that end, we apply $\mathcal{P}$ to a YES-instance G. We then use the labels applied by $\mathcal{P}$ to construct a NO-instance H for which $\mathcal{V}$ must return YES.

Our construction relies on a graph G, obtained from an undirected path by alternating the edge directions, and creating a cycle with the last two nodes (see Figure 3 for an illustration).

If the proof size is at most $\log\left(\frac{n-5}{2}\right)/2 - 1$ bits, then less than $\sqrt{n-5}/2\sqrt{2}$ different labels are available. Thus, in G a pair of adjacent labels A, B on the path will appear twice. Moreover, the nodes labeled A have only outgoing edges, and conversely, the nodes labeled B have only incoming edges. We obtain the acyclic NO-instance H by copying the pairs of nodes, and connecting them as depicted in Figure 3. This construction ensures that for all nodes u in H, there is a corresponding node v in G with $(\ell(u), D_2(u)) = (\ell(v), D_2(v))$. Therefore, the verifier $\mathcal{V}$ returns YES for all nodes in H.

Proof. Assume, for the sake of contradiction, there exists a D_2-PVP $(\mathcal{P}, \mathcal{V})$ for D-CYCLE using $\log\left(\frac{n-5}{2}\right)/2 - 1$ bits. Let G be the path $v_1, \ldots, v_{n-2}$ with $n - 2$ nodes and alternating edge directions, connected at v_{n-2} to the cycle v_n, v_{n-1} (which consists of just two nodes). The graph G is a YES-instance of

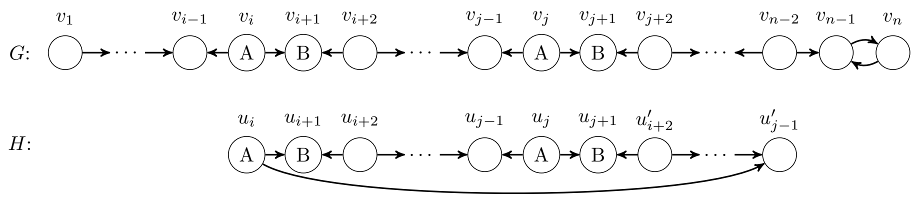

Figure 3: YES-instance G (with odd n) and NO-instance H of D-CYCLE. G consists of the n nodes $v_1, \dots, v_n$. For even k, node v_k has two incoming edges from v_{k-1} and v_{k+1}, whereas all v_k with odd k have two outgoing edges to v_{k-1} and v_{k+1}, i.e., the edge directions alternate. The prover $\mathcal{P}$ assigned the labels $\ell(v_i) = \ell(v_j) = \text{A}$ and $\ell(v_{i+1}) = \ell(v_{j+1}) = \text{B}$ to the corresponding nodes in G. In H, the nodes $u_{i+2}, \dots, u_{j-1}$ are copies of $v_i, \dots, v_{j+1}$ from G, and the nodes $u'_{i+2}, \dots, u'_{j-1}$ are obtained by copying (again) the nodes $v_{i+2}, \dots, v_{j-1}$. Note that H does not contain a cycle, but due to our construction each $u \in V(H)$ has a corresponding node $v \in V(G)$ with $(\ell(u), D_2(u)) = (\ell(v), D_2(v))$.

the problem, and thus the verifier $\mathcal{V}$ has to return YES for every node if the graph G was labeled by the prover $\mathcal{P}$.

We will now construct a NO-instance H (based on G and $\mathcal{P}$) such that for each $v_H \in V(H)$ there is a $v_G \in V(G)$ with $(\ell(v_H), D_2(v_H)) = (\ell(v_G), D_2(v_G))$, i.e., the verifier will output YES for every node in H.

First, we will prove that (due to the construction of G) there are $i \neq j$, with $2 \leq i, j \leq n-3$, such that *a)* $\ell(v_i) = \ell(v_j)$, *b)* $\ell(v_{i+1}) = \ell(v_{j+1})$, and *c)* $\text{dist}(v_i, v_j) = 2k, k \in \mathbb{N}$: There are at least $\lfloor \frac{n-5}{2} \rfloor$ pairs $(i, i+1)$ with $2 \leq i \leq n-3$ for each direction of the edge v_i, v_{i+1}. Labeling each pair differently requires at least $\sqrt{(n-5)/2}$ different labels, i.e., at least $\dfrac{1}{2}\log(\frac{n-5}{2})$ bits. Hence (using the pigeonhole principle), the claim holds.

The NO-instance H can now be constructed as follows: Let P be the (possibly empty) sub-path $v_{i+2}, \dots, v_{j-1}$ in G. We construct the cycle like structure H using two copies of P to connect copies of the pairs v_i, v_{i+1} and v_j, v_{j+1}, see Figure 3. We obtain the graph H with the nodes $u_i, \dots, u_{j+1}, u'_{i+2}, \dots, u'_{j-1}$, where the underlying undirected graph forms a ring.

It is left to show that we can assign labels to nodes in H such that $\mathcal{V}$ returns YES for all nodes in H. We assign the labels to the nodes in H by setting $\ell(u_x) = \ell(v_x)$ for all x and $\ell(u'_x) = \ell(v_x)$ for all x. It holds for each node $v_H \in H$ that there is a node v_G in G such that $D_2(v_H) = D_2(v_G)$ and $\ell(v_h) = \ell(v_G)$. Thus, as $\mathcal{V}$ returns YES for all nodes in G, $\mathcal{V}$ must return YES for all nodes in H, which contradicts that $(\mathcal{P}, \mathcal{V})$ is correct. $\square$

The claims (i) and (ii) of Theorem 1 for directed graphs are now established by Lemmas 2 to 4. The next two lemmas cover the undirected case (iii).

Lemma 5. *There is a U-PVP for* U-CYCLE *with a proof size of* 2 *bits.*

The upper bound for the optimal proof size is established by providing a *U*-PVP $(\mathcal{P}, \mathcal{V})$ with the desired proof label size. The idea is similar to the directed case, but this time the prover $\mathcal{P}$ labels all cycles with a 3 instead of a 0. Since removing all cycles from G leaves a forest of undirected trees (instead of the collection of DAGs in the directed case), one can save quite a few bits in the labels for the remaining nodes. For each tree, $\mathcal{P}$ picks a root node r that was originally adjacent to a cycle. In each tree, all nodes are labeled with their distance to r modulo 3. An example of a graph G labeled by $\mathcal{P}$ is depicted in Figure 4.

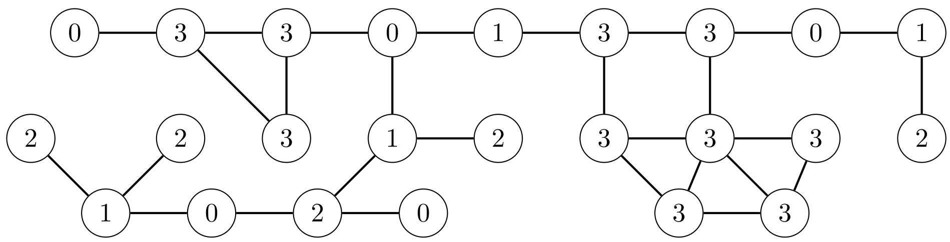

Figure 4: A labeled YES-instance of U-CYCLE. Nodes in cycles are labeled 3. The remaining nodes form a forest. After picking a root node adjacent to a cycle for each tree in the forest, all nodes in a tree are labeled with their distance (modulo 3) to the corresponding root.

The correctness of the PVP is established in a similar manner as in the directed case: The verifier $\mathcal{V}$ can then check if each node supposedly on a cycle (label 3) has at least two neighbors in a cycle, and if every other node (label $\neq 3$) has exactly one node "closer" to the root node of its tree. If G is acyclic, then there can be no node with label 3, as all nodes with a label of 3 would form a forest with at least one leaf. Assume for the sake of contradiction that the verifier returns YES for all nodes, and consider any node in some acyclic graph G. The path obtained by following the labels in descending order (modulo 3), i.e., going towards the root, must have infinite length, since there is no node adjacent to a cycle to break the succession.

Proof. We describe a U-prover-verifier pair $(\mathcal{P}, \mathcal{V})$ as required. Let $G = (V, E) \in$ U-CYCLE. The prover $\mathcal{P}$ labels all nodes $v \in V$ as follows: If v is part of a cycle, then $\ell(v) = 3$. By removing all nodes (and incident edges) that belong to a cycle, the graph decomposes into a set of Trees $\mathscr{T}$. Each tree $T \in \mathscr{T}$ is labeled by first picking a node $r \in T$ that was originally adjacent to a cycle and setting $\ell(r) = 0$. Then, for each other node $t \in T$ let $\operatorname{dist}_T(r, t)$ be the distance from r to t in T and set $\ell(t) = \operatorname{dist}_T(r, t) \mod 3$. An example for the labeling can be found in Figure 4. As only the labels $\{0, 1, 2, 3\}$ are used, 2 bits suffice.

The verifier $\mathcal{V}$ returns YES for nodes v with a) at least two neighbors have a label of 3 if $\ell(v) = 3$ or b) if $\ell(v) = j$, $j \in \{0, 1, 2\}$, then the following three conditions must be fulfilled: i) There is no neighbor with a label of j, ii) There is exactly one neighbor with a label of $j - 1$ if $j \in \{1, 2\}$ or at most one neighbor with a label of 2 if $j = 0$, and iii) all other neighbors must have a label of exactly $j + 1 \mod 3$ or 3. In all other cases, $\mathcal{V}$ returns NO. If a node v is part of an undirected cycle (hence, $\ell(v) = 3$), then it has at least two neighbors in the cycle with the label 3, meaning that $\mathcal{V}$ returns YES for v. Else, consider the tree $T \in \mathscr{T}$ from above with $v \in T$ with the corresponding "root" node r picked by the prover. If $v = r$, then all neighbors in T have the label 1 and all other neighbors (of whom at least one exists) are on cycles with a label of 3. Thus, $\mathcal{V}$ outputs YES for $v = r$. If $v \in T$ and $v \neq r$, then all neighbors v' of v in T are labeled according to $\ell(v') = \mathrm{dist}_T(r, v') \mod 3$. All other neighbors (if any exist) of v in G must be on cycles with a label of 3. Hence, $\mathcal{V}$ returns also YES in this case.

For the U-prover-verifier pair $(\mathcal{P}, \mathcal{V})$ to be correct, it is left to show that $\mathcal{V}$ returns NO for at least one node if the considered graph is not in U-CYCLE. Let G_{no} be a connected undirected graph containing no cycle.

Assume there would be a node $v \in V(G_{no})$ with $\ell(v) = 3$. Consider all nodes with a label of 3 in $V(G_{no})$: As there is no cycle, the subgraph(s) induced by these nodes form a forest $\mathscr{F}$. Let $T \in \mathscr{F}$ be the tree with $v \in T$. Pick a leaf of T: It has at most one neighbor with a label of 3, meaning that $\mathcal{V}$ will return NO for at least one node.

Thus, no node v with $\ell(v) = 3$ can exist. Now, pick any node $v \in V(G_{no}$ with $\ell(v) \in \{0, 1, 2\}$. Node v (and also any other node in $V(G_{no})$) must have exactly one neighbor v_1 with a label of $\ell(v_1) = \ell(v) - 1 \mod 3$, as else $\mathcal{V}$ would return NO for v. Consider the path starting from v that picks as its next node the unique neighbor with a label smaller by one modulo 3, i.e., $v, v_1, \ldots$ - until no such node exists any more. Since G_{no} is cycle-free, the path must be finite and end at some node v_j. As v_j has no neighbor with a label of 3 or a label of $\ell(v_j) - 1 \mod 3$, the verifier $\mathcal{V}$ returns NO for v_j. Thus $\mathcal{V}$ will return NO for any connected graph not containing a cycle, meaning that the U-PVP $(\mathcal{P}, \mathcal{V})$ is correct. $\square$

Lemma 6. *The U-PVP proof size for* U-CYCLE *is at least* 2 *bits.*

Proof. The proof is by case distinction. Assume there exists a U-PVP $(\mathcal{P}, \mathcal{V})$ for U-CYCLE using 1 bit. We use a YES-instance $G = (V(G), E(G))$ of U-CYCLE consisting of a cycle with three nodes with a path P of three nodes attached to it. We will show that for any labeling ℓ assigned to the nodes on the path P, for which $\mathcal{V}$ returns YES for all nodes in G, there exists a NO-instance $H = (V(H), E(H))$ of U-CYCLE for which $\mathcal{V}$ must also return YES for all nodes in H. W.l.o.g. consider the four cases in Figure 5.

These four cases combined with their analogous inversions, where all labels are switched on the path P, present all combinations of how labels can be assigned to the nodes on the path P. For every YES-instance G there exists a NO-instance H such that for each $v_H \in V(H)$ there is a $v_G \in V(G)$ with $(\ell(v_H), U(v_H)) = (\ell(v_G), U(v_G))$. Since $\mathcal{V}$ can not differentiate between v_H and V_G, it must also return YES for all nodes in the corresponding NO-instance, which contradicts that $(\mathcal{P}, \mathcal{V})$ is correct. It follows that there is no correct proof labeling scheme $(\mathcal{P}, \mathcal{V})$ using only 1 bit. $\square$

We note that one could deviate from the notion of studying the label size in bits, by studying how many different labels are sufficient and necessary. E.g., for U-CYCLE, we showed that 1 bit does not suffice, but 2 bits are sufficient, raising the question of studying three different labels, using $\approx 1.58\ldots$ bits at

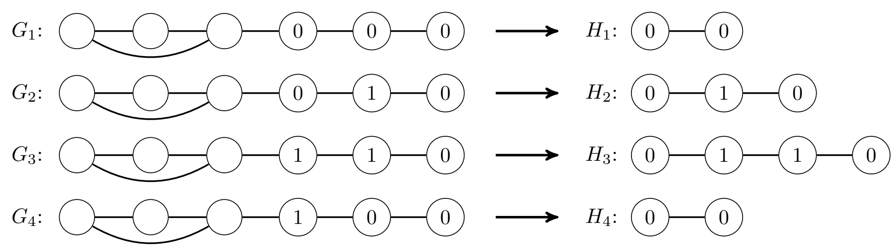

Figure 5: YES-instances G_1, G_2, G_3, G_4 and NO-instances H_1, H_2, H_3, H_4 of U-CYCLE. For any labeling assigned to the YES-instances, there exists a NO-instance for which $\mathcal{V}$ must return YES for all nodes. In these graphs, the labels for the nodes in the cycle can be chosen arbitrarily. The numbers in the remaining nodes are their labels. All labels in this figure can be inverted to get the remaining 4 possible combinations for a labeling.

every node. However, the lower bound idea from the proof of Lemma 6 does not extend to three labels, and neither does the method from the 2-bit PVP for U-CYCLE. We believe that intricate techniques beyond those used in this article will be needed to settle the question of three labels for U-CYCLE.

2.2. Acyclicity

In the undirected case, an acyclic graph is nothing but an undirected tree. The question of detecting undirected trees was already answered in [35] (see Section 2.3). In the directed case, however, not every acyclic graph is necessarily a tree. Let D-ACYCLIC denote the set of all weakly connected directed acyclic graphs. In the remainder of this section we establish the following:

Theorem 7. *For the acyclicity detection problem, it holds that*
(i) The D_1-proof size for D-ACYCLIC *is $\Theta(\log n)$ bits.*
(ii) The D_2-proof size for D-ACYCLIC *is $\Theta(\log n)$ bits.*

While not every directed acyclic graph is a directed tree, the converse holds, i.e., every directed tree is a directed acyclic graph. Techniques similar to those used by Korman et al. [35] can be used to obtain the claimed lower bound for tree detection in our model. Hence, we only need to establish the upper bounds in Theorem 7. Note that any D_1-PVP immediately yields a D_2-PVP with the same proof size by simply ignoring the information obtained via outgoing edges. It is therefore sufficient to find a D_1-PVP with the desired proof size.

Lemma 8. *There is a D_1-PVP for* D-ACYCLIC *with a proof size of* $\log n$ *bits.*

In the proof, the prover assigns each node with no incoming labels the label 0, and each other node the highest incoming label plus one. We refer to Figure 6 for illustration.

Thus, each node with label $j > 0$ can check if there is an incoming label $j-1$, or when $j = 0$, if the multiset of incoming labels is the empty set. As each NO-instance contains a cycle, a node with the highest label in the cycle would send its label to another node, causing this node to output NO.

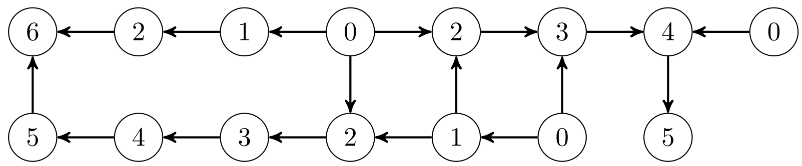

Figure 6: A labeled Yes-instance of D-Acyclic. Nodes without incoming edges are labeled 0, all other nodes have a label that is equal to the highest incoming label plus 1.

Proof. We describe a D_1-prover-verifier pair $(\mathcal{P}, \mathcal{V})$ as required. Let $G = (V, E) \in$ D-Acyclic and let $V_0 \subseteq V$ be the set of all nodes $v_0 \in V$ with $D_1(v) = [\,]$, i.e., v_0 has zero incoming edges. The prover $\mathcal{P}$ labels all nodes $v \in V$ as follows. *a*) All nodes $v_0 \in V_0$ have the label $\ell(v_0) = 0$, and *b*) for all other nodes $v_+ \in V$ holds: $\ell(v_+) = 1 + \max_{(u,v_+)\in E} \ell(u)$. We refer to Figure 6 for an example. As a label i requires a label $i - 1$ to exist, the highest label is bounded from above by n, inducing a maximum label size of $\log n$ bits.

The verifier $\mathcal{V}$ returns Yes for nodes v with *a*) $D_1(v) = [\,]$ if $\ell(v) = 0$ or *b*) $\ell(v) = 1 + \max_{(u,v)\in E} \ell(u)$ if $D_1(v) \neq [\,]$. In all other cases, $\mathcal{V}$ returns No. Thus, the verifier returns Yes for all nodes in V if G was labeled by $\mathcal{P}$, as all incoming labels are available to the verifier.

For the D_1-prover-verifier pair $(\mathcal{P}, \mathcal{V})$ to be correct, it is left to show that $\mathcal{V}$ returns No for at least one node if the considered graph is not in D-Acyclic. Let G_c be a weakly connected directed graph containing a directed cycle $C = v_1, v_2, \ldots, v_{|C|}, v_1$. W.l.o.g., let $v_i \in C$ be a node with the highest labeling in C. Consider the outgoing edge from v_i in C: The corresponding neighbor of v_i in C cannot have a higher label than v_i. Thus $\mathcal{V}$ will return No, meaning that the D_1-PVP $(\mathcal{P}, \mathcal{V})$ is correct. $\square$

2.3. Trees

Let U-Tree denote the set of all undirected trees. Let correspondingly D-Tree denote the set of all weakly connected directed trees in which all edges are directed away from some unique root node.

Theorem 9 ([35])**.** *For the tree detection problem, it holds that*

(i) The proof size for U-Tree *is* $\Theta(\log n)$ *bits.*

(ii) The D_1*-proof size for* D-Tree *is* $\Theta(\log n)$ *bits.*

(iii) The D_2*-proof size for* D-Tree *is* $\Theta(\log n)$ *bits.*

While the authors of [35] assumed port numbers to be available, the PLS used in the upper bound construction do not make use of them. Therefore, the upper bound claims (i) and (ii) carry over to our model. Since their port numbering model is strictly stronger than ours, the same is true for the lower bounds. Naturally, upper bounds for D_1-proof sizes carry over to the D_2-case, so the only thing that is left is to show that there exists no D_2-PVP with a proof size of $o(\log n)$ bits. Since the counter example construction to establish this claim are very similar to the construction used in [35], we omit the details here.

3. Port Numbers vs. s-t Reachability

As pointed out by Göös and Suomela in [29], "*To ask meaningful questions about connectivity [...] we have the promise that there is exactly one node with label s and exactly one node with label t.*" In this section, we thus assume that all graphs have at least two nodes, of which one node has the unique label s and another node has the unique label t. It is known that in the undirected case, the U-proof size for s-t reachability is 1 bit, see Section 1. In the directed case, on which we focus, a non-trivial lower bound remained an open question [29]. For that, let s-t REACHABILITY denote the set of all directed graphs containing a directed path from s to t.

We show a lower bound for s-t REACHABILITY with one-way communication by combining our previously used techniques. The upper bound for the two-way case requires a new insight: As it turns out, port numbers can be emulated in our model by implementing a 2-hop coloring with only $O(\log \Delta)$ bits[4]. Then, whenever a port number is required for some proof, we only need to pay at most $O(\log \Delta)$ bits. While this seems like a high price to pay, we note that referring to a specific port number requires $O(\log \Delta)$ bits even if the port numbering itself is provided for free. We will later see how this applies in the case of two-way s-t REACHABILITY (cf. [29]). In the remainder of this section we establish the following theorem:

Theorem 10. *For the s-t REACHABILITY problem, it holds that*

(i) The D_1-proof size for s-t REACHABILITY is $\Theta(\log n)$ bits.

(ii) The D_2-proof size for s-t REACHABILITY is at most $O(\log \Delta)$ bits.

To see that s-t REACHABILITY permits a D_1-PVP with a proof size of $O(\log n)$, observe that the nodes on the path can simply be enumerated, cf. Figure 7.

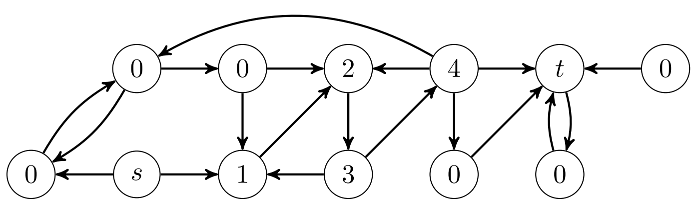

Figure 7: A labeled YES-instance of s-t REACHABILITY. All nodes v in G on the shortest s-t path P of length 5 are labeled $\ell(v) = \mathrm{dist}(s, v)$. All other nodes are labeled 0.

Each node on the path can now check whether it has a predecessor on the path, i.e., every YES-instance is verified correctly. To see that NO-instances will be rejected, one can follow a similar line of arguments as in Lemma 8: Every path obtained by following descending incoming labels, starting from t, must

[4]For more applications of 2-hop colorings in anonymous networks we refer to the recent article of Emek et al. [17].

end in a node without a predecessor, since the graph is finite and s and t are not connected.

Lemma 11. *There is D_1-PVP for s-t* REACHABILITY *with a proof size of $O(\log n)$ bits.*

Proof. We describe a D_1-prover-verifier pair $(\mathcal{P}, \mathcal{V})$ as required. Let the directed graph $G = (V, E) \in s\text{-}t$ REACHABILITY and let $P = s, v_1, \ldots, v_j, t$ be a shortest directed path from s to t. The prover $\mathcal{P}$ labels all nodes $v \notin P$ with $\ell(v) = 0$ and each node $v_i \in P = s, v_1, \ldots, v_j, t$ with $\ell(v_i) = i$, i.e., $\ell(v_i) = \operatorname{dist}(s, v_i)$ by definition of P. We refer to Figure 7 for illustration. As $\operatorname{dist}(s,t) \leq n$, $\ell(s) = s$, and $\ell(t) = t$, the proof size is in $O(\log n)$ bits.

The verifier $\mathcal{V}$ returns YES for all nodes v with a label of 0 and for the node s with unique label $\ell(s) = s$. For the node t with the unique label $\ell(t) = t$, YES is returned if *a*) the label s is received, or *b*) if a label greater than zero is received. $\mathcal{V}$ returns YES for all other nodes v with label $\ell(v) = i > 1$, if one of the received labels is $i - 1$. For the special case of $\ell(v) = 1$, one of the received labels has to be s.

Thus, the verifier will return YES at all nodes for YES-instances labeled by $\mathcal{P}$: The nodes v with $\ell(v) = 0$ and s return YES. Furthermore, as each other node is on the path $P = s, v_1, \ldots, v_j, t$ with $\ell(v_i) = i$, they have a predecessor on the path with the desired label, and hence return YES as well.

It is left to show that $\mathcal{V}$ returns NO for at least one node if the graph G is not in s-t REACHABILITY. Let H be a NO-instance of s-t REACHABILITY, i.e., there is no directed path from s to t. Consider the set Z of nodes that can be reached from t by traversing directed edges in the reverse direction to a node with a label lower by exactly one, or in the case of t, with any label greater than zero. Note that by definition of H, there is no node $v' \in Z$ such that there is an edge $(s, v') \in H(E)$. Let v^* be a node with the lowest label $\ell(v^*) = x$ in Z: As v^* cannot receive a label $x - 1$ or the label s, v^* will return NO. Hence, the described D_1-PVP $(\mathcal{P}, \mathcal{V})$ is correct. □

Lemma 12. *The D_1-PVP proof size for s-t* REACHABILITY *is at least* $\log\left(\frac{n}{4}\right) - 2$ *bits.*

Proof. Assume, for the sake of contradiction, that there is a D_1-PVP $(\mathcal{P}, \mathcal{V})$ for s-t REACHABILITY with a proof size of $\log(n/4) - 3$ bits. Let n be odd and let G be the directed path $P = v_1, \ldots, v_n$ where $v_1 = s$, and $v_n = t$. We add to G directed edges so that all nodes v_k with $n > k > \lceil n/2 \rceil$ have an outgoing edge to $v_{k - \lceil n/2 \rceil + 1}$, as depicted in Figure 8.

We note that G is a YES-instance for s-t REACHABILITY, and that there is only one simple path from s to t in G. Like above, we now apply $\mathcal{P}$ to G and use the obtained labels ℓ to construct a NO-instance H with a labeling ℓ'. The construction ensures that for every node u in H, there is a node v in G with $(\ell'(u), D_1(u)) = (\ell(v), D_1(v))$.

With an argument analogous to that in the proof of the previous Lemma 4, we will first show that there are $i \neq j$, with $\lceil n/2 \rceil + 2 \leq i \leq n - 4$ and $i + 2 \leq$

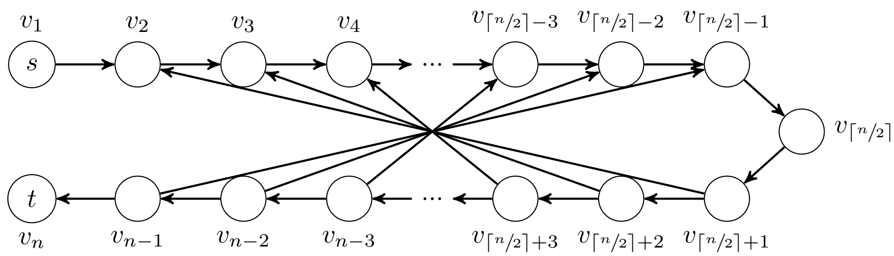

Figure 8: The YES-instance G of s-t REACHABILITY used in our proof of Lemma 12. Nodes v_j with $j > \lceil n/2 \rceil$ have an outgoing edge to $v_{j-\lceil n/2 \rceil+1}$. Note that G contains only one simple s-t path.

$j \leq n-2$, such that $\ell(v_i) = \ell(v_j)$. In other words, we are looking for two non-adjacent nodes v_i and v_j that are within the second half of the path P, and v_i comes before v_j on P. Suppose that there are no such v_i, v_j. Consequently, $\ell(v_{\lceil n/2 \rceil+2})$ must be different from the label of each node in $\{v_{\lceil n/2 \rceil+4}, \ldots, v_{n-2}\}$. By induction, if v_i, v_j with the desired properties do not exist, then there need to be at least $\lfloor \frac{n}{4} \rfloor - 2$ different labels on the sub-path $v_{\lceil n/2 \rceil+2}, \ldots, v_{n-2}$. This is a contradiction to the assumption that the proof size is limited to $\log(n/4) - 3$ bits, and we conclude that such nodes v_i, v_j must be present.

To complete our proof of Lemma 12, we now construct the (weakly connected) NO-instance H and the labeling ℓ'. For that, let v_i and v_j be the two nodes in G with $\ell(v_i) = \ell(v_j)$ as above. We denote by x the node $v_{j-\lceil n/2 \rceil+1}$ in the first half of P with an incoming edge from v_j. To construct H and ℓ', we first copy the graph G including the labels assigned by ℓ. We then replace the edges (v_i, v_{i+1}) and (v_j, x) by (v_i, x) and (v_j, v_{i+1}) (see Figure 9). Note that in H, the two distinguished nodes s and t are no longer connected by a directed path.

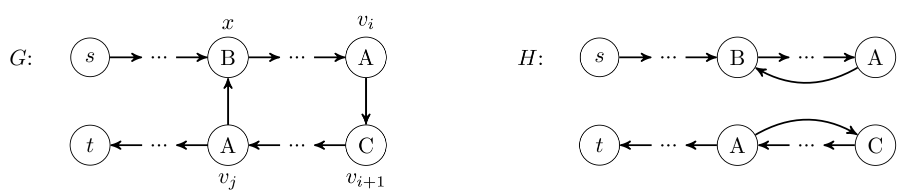

Figure 9: YES-instance G of s-t REACHABILITY labeled by $\mathcal{P}$, and the corresponding NO-instance H for s-t REACHABILITY. Some label A appears twice on the s-t path, namely at the nodes v_i and v_j. Since v_j is at least two steps after v_i and has an outgoing edge to a node x before v_i, the NO-instance H for which $\mathcal{V}$ fails can be constructed. For the sake of simplicity not all edges are shown.

It is left to show that $\mathcal{V}$ will return YES for all nodes in H when labeled with ℓ'. Note that in H, the only nodes that were changed in some way in comparison to G were v_i, v_{i+1}, v_j and x. However, all four nodes still have the same labels, and the incoming edges were changed only for x and v_{i+1}. As it holds that $\ell'(v_i) = \ell'(v_j)$, it follows that $D_1(v_{i+1})$ is the same in G and in H, equivalently for $D_1(x)$. Thus, since the verifier $\mathcal{V}$ returned YES for all nodes in G, $\mathcal{V}$ must also return YES for all nodes in H, contradicting that $(\mathcal{P}, \mathcal{V})$ is correct. $\square$

We note that the construction in the proof of Lemma 12 has constant degree in every node. Therefore, there cannot be a D_1-PVP for which the proof size depends only on Δ. The missing part to establish Theorem 10 is a D_2-PVP for s-t REACHABILITY.

The PVP for this problem as proposed by [29] relies on the nodes' ability to "point to an edge" by using its port number. The authors suggest to mark an s-t path P by simply pointing to the edges used by P. We argue that one can point to an edge in our models U and D_2, even though port numbers are not available. To that end, we enrich the labels to include a 2-hop coloring of the graph G, i.e., a coloring (node labeling) such that the label of each node v is unique among all nodes with distance at most 2. We denote this coloring by $c(v)$.

Since a 2-hop coloring requires at most $\Delta^2 + 1$ colors, each color can be encoded using $O(\log \Delta)$ bits. Moreover, the 2-hop coloring can be checked locally, since each node v only needs to verify if v and all its neighbors have different colors. A PVP can now rely on the fact that v's color is unique among u's neighbors.

To obtain a D_2-PVP for for s-t REACHABILITY, a node $u \in V$ can now point to an edge $(u, v) \in E$ by referencing $c(v)$ in its label. In this way, the pointed-to edge is uniquely specified for both u and v. Applying the same reasoning as in [29], we obtain the following lemma, which together with Lemmas 11 and 12 concludes our effort to establish Theorem 10.

Lemma 13. *There is a D_2-PVP for s-t REACHABILITY with a proof size of $O(\log \Delta)$ bits.*

4. s-t Reachability: Beyond a Single Round of Communication

In the last section, we combined our developed techniques to obtain the first non-trivial lower bound for the label size of directed one-way s-t reachability that does not rely on descriptive complexity methods. In this section, we will show how our lower bound can be extended to models where more than a single round of communication is allowed. This idea is inspired by the model of Locally Checkable Proofs by Göös and Suomela [29], where labels may be exchanged over a constant number of communication rounds.

To be more precise, we will show that relaying the labels of the in-neighbors over k rounds (i.e., *k rounds of communication*, with $1 \leq k \leq n$) reduces our lower bound on the label size proportionately to k. More formally, the following holds:

Lemma 14. *The D_1-PVP proof size for s-t REACHABILITY with k rounds of communications is at least $\Omega\left(\frac{1}{k}\log n\right)$ bits.*

Proof. We essentially use the same construction as in the proof of Lemma 12, but with one modification: In the graph used (cf. Figure 8) we replace each of the $\lceil n/2 \rceil - 2$ directed edges between the nodes $v_{\lceil n/2 \rceil+1}, \ldots, v_{n-1}$ and $v_2, \ldots, v_{\lceil n/2 \rceil -1}$ with directed paths of k nodes. The new graph G' has thus $n' = n + (\lceil n/2 \rceil - 2)\, k \in \Theta(nk)$ nodes.

To obtain the result of Lemma 14, we need to bound the number of bits x of the label size s.t. there are two of these paths $v_1^1, \ldots, v_k^1$ and $v_1^2, \ldots, v_k^2$ of length k in G' with exactly the same sequence of labels $\ell(v_1^1) = \ell(v_1^2), \ldots, \ell(v_k^1) = \ell(v_k^2)$. Thus, we will study the size of x s.t. two labels of length kx exist among $\lceil n/2 \rceil - 2$ nodes. I.e., for what x does $2^{kx} < \lceil n/2 \rceil - 2$ hold? For our purposes, $2^{kx} < n/2 - 2$ is sufficient. We obtain $kx < \log(n/2-2)$, resulting in a weakened bound of $x < \frac{\log n}{k} - \frac{\log 2}{k} - \frac{2}{k} \in \Omega\left(\frac{\log n}{k}\right)$.

As the number of nodes n' in G' is $\Theta(nk)$, $n \in \Theta(n'/k)$ holds. Due to logarithmic identities, $\Omega\left(1/k \log\left(\frac{n'}{k}\right)\right) \in \Omega\left(\frac{\log n'}{k}\right)$, yielding Lemma 14. □

Hence, our results carry over for the model where a constant number of communication rounds is allowed.

Theorem 15. *Let k be any constant. The D_1-proof size for s-t* REACHABILITY *with k rounds of communications is $\Theta(\log n)$ bits.*

For an upper bound beyond constant communication range, the construction for the proof of Lemma 11 can be modified as well, by labeling the nodes along a shortest directed s-t path P with a label size of $O(\log\left(\frac{n}{k}\right))$: The prover labels the first k nodes on P after s with 1, the k next nodes with 2, and so on. The verifier can be left nearly identical We omit the details of the analogous proof:

Lemma 16. *There is D_1-PVP for s-t* REACHABILITY *with k rounds of communications with a proof size of $O(\log\left(\frac{n}{k}\right))$ bits.*

We note that there is still a gap between the upper and lower bounds of Lemma 14 and 16. An idea to close this gap would be to distribute the $\log n$-size numbers over k nodes, obtaining an upper bound of $O(\frac{1}{k}\log n)$. However, even keeping techniques such as padding and marking the "beginning/end" of a number in mind, an adversary can place back edges from nodes representing numbers closer to t to a set of nodes representing some number i, tricking them into believing they are predated by some non-existent number $i-1$. We believe however, that this idea might be of use in stronger models with "more strings attached", such as (incoming) port numbers or unique identifiers.

5. Application to Network Updates in Software Defined Networks

While the current line of research on local checkability focuses extensively on its theoretical properties, the number of practical applications beyond verification of a proof is sparse to the best of our knowledge. E.g., as mentioned in the related work Subsection 1.1, Schmid and Suomela [44] use proof labeling schemes to verify spanning trees in networks. They make use of a recent development in networking, so-called Software Defined Networks (SDNs) (cf., e.g., [14]), which resembles the model of local checkability in some sense: In SDNs, a (logically) centralized controller oversees the switches in the network itself, updating their behavior on how to deal with data packets. As such, the control and data plane of networks are decoupled, with the controller becoming the prover and the switches becoming the verifiers.

In this section, we will extend their line of work beyond the verification of spanning trees, to the consistent migration between spanning trees. We will use the notion of routing trees in this context, as their intended use is to perform routing to a fixed root of all trees. As such, a migration via updates is called consistent, if the spanning tree property is always maintained. I.e., no (temporary) forwarding loops are introduced in the process. More formally:

Let $G = (V, E)$ be a connected directed graph, with every edge (u, v) also having a back edge (v, u).[5] Let $T_1 = (V, E_1)$ and $T_2 = (V, E_2)$ be directed trees with a common root node $r \in V$ and $E_1, E_2 \subset E$. A node $v \in V, v \neq r$ changing its parent from the one in T_1 to the one in T_2 is called an *update*. An update is called *consistent* if the resulting graph is a directed tree rooted in r. A sequence of consistent updates resulting in T_2 is called a consistent *migration*.

We will first discuss the current state of the art of consistent migration between spanning trees in SDNs with a centralized controller in Subsection 5.1 and compare it with the paradigm of local checkability, before proposing a local migration scheme in Subsection 5.2.

5.1. Consistent Migration between Routing Trees in SDNs

With the central controller having the power to change the switches' behavior for optimization purposes, the network itself is in a constant state of change, several hundred times per day in practice [31, 32]. As thus, another problem surfaces: The (temporal) breaking of the verified properties. Imagine all switches change from a routing tree T_1 for a destination r to another routing tree T_2. If a node u is the parent of v in T_1, but v is a parent of u in T_2, then u updating before v temporarily breaks the tree property, inducing a loop, cf. Figure 10.

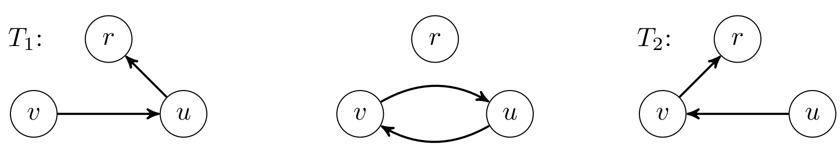

Figure 10: In the tree T_1 to the left, v forwards all packets for r to u, which in turn forwards all packets to r. This situation is flipped in T_2: u forwards all packets for r to v, which in turn forwards all packets to r. Should u update before v, then the situation in the middle occurs: The graph is no longer a tree, and all packets loop between v and u.

[5] I.e., we consider standard full-duplex networks.

This asynchronous behavior is not a technicality, switches can take different times to apply updates (especially under load), or might even temporarily not be available to the controller [30, 33, 36]. Hence, it is not possible to guarantee two updates taking place at exactly the same time, inducing an ordering of all updates. However, the controller may send out multiple updates in parallel, this subset of updates could be executed in any order though.

To guarantee consistency, the authors of [3, 19, 21, 38, 40, 46] pre-compute a (partial) update order, send out the individual updates (possibly multiple at once), and only proceed with the next set of updates if the previous one is confirmed. E.g., in the previously discussed example depicted in Figure 10, v could update before u, always maintaining the tree property in the network. This involves an ongoing interaction with the central controller, unlike the local checkability paradigm, in which the prover sends out the proofs once, and is only involved again if the verification fails.

The methods of [3, 19, 21, 38, 40, 46] are inherently non-local however: Their algorithms check for updates that will not violate the tree property, but these updates can be anywhere in the network. Consider the example in Figure 11, where with the previously mentioned methods, the whole network can be updated in a few rounds of switch-controller interaction, but the nodes with new forwarding rules cannot decide with local communication[6] whether it is safe to update or not.

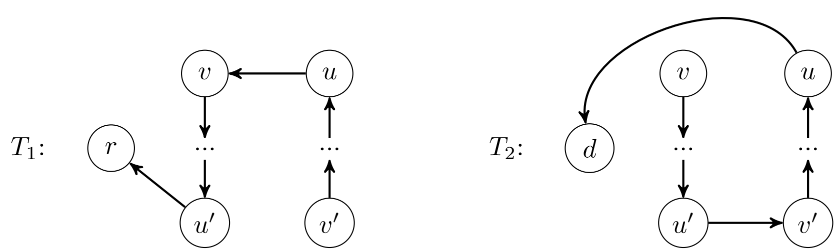

Figure 11: The consistent migration from T_1 to T_2 can be handled in two updates by a centralized controller: First, u updates, and then after u confirmed the update to the controller, in a second update, the controller tells u' to update. As the distance between the nodes u, v and u', v' can be $\Omega(n)$, non-local communication via necessary: Else, u' cannot know when it is safe to update without inducing a loop.

A different approach is taken in [27, 28], which can be seen as analogous to the local checkability approach for acyclicity and trees: For the new tree T_2, label the root with (update) 0, then its children with (update) 1, and so on, labeling each node with an (update) number equivalent to its distance to the root of T_2. As T_2 could have a depth of $n-1 \in \Omega(n)$, they need $\Omega(n)$ (or depth of T_2) subsequent updates in the worst case. No faster algorithm (dependent on n or the tree depth) can exist either for the general case: Consider a degenerated

[6] I.e., in a constant number of rounds.

tree T_1, where every node has at most one child. If the parent-child relation is flipped in T_2 (with the only leaf becoming the only child of the root), then $\Omega(n)$ (or depth of T_2) updates are required, cf., e.g., [38].

5.2. A Local Migration Scheme

As pointed out in Figure 11, the methods of, e.g., [19, 38, 40, 46], are not applicable for a local migration scheme. Thus, we will extend the scheme used by [27, 28], which has the same worst-case number of updates needed in SDNs. Every node should be able to decide locally if it can change its forwarding behavior to the new routing tree T_2.

We note that the nodes cannot verify if the new tree T' is actually T_2, as the very nature of T_2 is decided upon by the central controller. An analogous case can be about identifiers, which is why we will adopt the model of Schmid and Suomela [44] and assume every node has a unique identifier $\texttt{id}(v)$ (of size $O(\log n)$). As thus, we will only make sure that the nodes update to the tree specified by the sent out labels. We will still maintain a tree property at all times, no matter what labels are sent by the controller.

Algorithm 17 (From the perspective of a node v)**.**
Situation: Graph $G = (V, E)$ with the forwarding rules of the nodes forming a directed tree $T_1 = (V, E_1)$ with root $r \in V$. Every node $v \in V, v \neq r$ gets a label consisting of $\texttt{id}(p_2(v))$ of its parent $p_2(v)$ in T_2 and the depth $d_2(v)$ of v in T_2.

If $p_2(v)$ sends depth $d_2(p_2(v)) = d_2(v) - 1$ or $p_2(v) = d$, and v didn't update yet:
Update forwarding rule to $p_2(v)$ and then send $d_2(v)$ to all neighbors.

Lemma 18. *Algorithm 17 takes at most $n - 1$ updates.*

Proof. r will never update, and every other node will update at most once. $\square$

We will now prove that Algorithm 17 works as intended, if the labels are correct:

Lemma 19. *Let the situation described in Algorithm 17 be correct: Then, executing Algorithm 17, the nodes $v \in V$ perform consistent updates, leading to a consistent migration from T_1 to T_2.*

Proof. We will first prove that every update is consistent, before showing that a consistent migration to T_2 occurs.

Observe that initially, there is no loop in the network. Furthermore, note that during the whole execution of Algorithm 17, every node v has either a forwarding rule pointing at its parent $p_1(v)$ in T_1 or a forwarding rule pointing at its parent $p_2(v)$.

Assume for the sake of contradiction that the update of the node v' is the first occurrence of a loop in Algorithm 17. W.l.o.g., let this loop be $v' = v_0, v_1, v_2, \ldots, v_k, v' = v_{k+1}$. v_0 will only have updated to v_1 after v_1 has updated to v_2, with v_1 only updating to v_2 after v_2 has updated to v_3, and so on. As thus, for $1 \leq i \leq k+1$, v_i must be the parent of v_{i-1} in T_2. This leads to the desired contradiction: As T_2 is a tree, no loop $v' = v_0, v_1, v_2, \ldots, v_k, v' = v_{k+1}$ can exist in it. Hence, every update performed by Algorithm 17 is consistent.

As every update is consistent, Algorithm 17 performs a consistent migration, but it is left to show that the migration will reach T_2. Now, assume, again for the sake of contradiction, that at some point no node can update any longer, but the forwarding rules do not form T_2. If every node (except for r) has updated, then the forwarding rules form exactly T_2. Thus, let $v' \neq r$ be a node which has not updated yet. We can use similar reasoning to the paragraph above: If v' has not updated yet, then $p_2(v')$ has not updated yet, which means in turn $p_2(p_2(v'))$ has not updated yet, and so on. As the graph is finite, this leads to the desired contradiction, as else a loop would need to exist. □

However, the network could still end up in a loop if the new forwarding rules do not form a tree T_2, but contain some loop, due to the error of the controller. As we incorporated the depth of each node into its label, this will be prevented:

Lemma 20. *Let the situation described in Algorithm 17 be correct, except that the new forwarding rules do not form a tree. Then, the updates performed by Algorithm 17 will still be consistent and not induce a loop.*

Proof. W.l.o.g., assume for the sake of contradiction that the update of a node v' is the first update of Algorithm 17 inducing a loop $v' = v_0, v_1, v_2, \ldots, v_k, v' = v_{k+1}$. As shown in the proof of Lemma 19, this loop must be exclusively induced by the new forwarding rules, which could be the case now as the new forwarding rules no longer have to form a tree. However, when updating, every node also checks if the label for the depth of the tree is smaller than its own by exactly one. Consider the smallest depth given as part of a labeling to a node v_i in the loop $v' = v_0, v_1, v_2, \ldots, v_k, v' = v_{k+1}$. When v_i updated, it checked the depth of its parent to be exactly one smaller than its own. However, as v_i has the smallest depth in the loop, this is a contradiction: v_i would not have updated. □

5.3. Further Applications in SDNs

The loop freedom of forwarding rules is just one of many consistency properties to be considered when performing changes in the behavior of switches of Software Defined Networks via the controller, cf. [20]. We envision that local updates for other consistency properties can be developed as well, e.g., for black hole freedom [19], per-packet consistency [15, 42], waypoint enforcement [37, 39, 46], and bandwidth capacity constraints for network flows [12, 13, 31, 33].

Acknowledgments

We would like to thank the anonymous reviewers of the 17th International Conference on Distributed Computing and Networking (ICDCN '16) for their helpful comments on our preliminary extended abstract [18].

Furthermore, we would like to thank the anonymous reviewers of this journal article for their helpful comments as well, among many other things they revealed a (now fixed) error in Lemma 16, suggested to write $\log n$-bit number distributed over k nodes in this context, and pointed out that one could study the exact proof size beyond just the size in bits.

Lastly, Klaus-Tycho Foerster is at Microsoft Research, but most of the work was done at ETH Zurich, supported in part by Microsoft Research.

References

[1] Yehuda Afek, Shay Kutten, and Moti Yung. The local detection paradigm and its application to self-stabilization. *Theor. Comput. Sci.*, 186(1-2):199–229, 1997.

[2] Miklós Ajtai and Ronald Fagin. Reachability is harder for directed than for undirected finite graphs. *J. Symb. Log.*, 55(1):113–150, 1990.

[3] Saeed Amiri, Arne Ludwig, Jan Marcinkowski, and Stefan Schmid. Transiently consistent sdn updates: Being greedy is hard. In *Structural Information and Communication Complexity - 23rd International Colloquium, SIROCCO 2016, Helsinki, Finland*, 2016.

[4] Heger Arfaoui, Pierre Fraigniaud, David Ilcinkas, and Fabien Mathieu. Distributedly testing cycle-freeness. In Dieter Kratsch and Ioan Todinca, editors, *Graph-Theoretic Concepts in Computer Science - 40th International Workshop, WG 2014, Nouan-le-Fuzelier, France, June 25-27, 2014. Revised Selected Papers*, volume 8747 of *Lecture Notes in Computer Science*, pages 15–28. Springer, 2014.

[5] Heger Arfaoui, Pierre Fraigniaud, and Andrzej Pelc. Local decision and verification with bounded-size outputs. In Teruo Higashino, Yoshiaki Katayama, Toshimitsu Masuzawa, Maria Potop-Butucaru, and Masafumi Yamashita, editors, *Stabilization, Safety, and Security of Distributed Systems - 15th International Symposium, SSS 2013, Osaka, Japan, November 13-16, 2013. Proceedings*, volume 8255 of *Lecture Notes in Computer Science*, pages 133–147. Springer, 2013.

[6] Baruch Awerbuch, Boaz Patt-Shamir, and George Varghese. Self-stabilization by local checking and correction (extended abstract). In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 268–277. IEEE Computer Society, 1991.

[7] Baruch Awerbuch, Boaz Patt-Shamir, George Varghese, and Shlomi Dolev. Self-stabilization by local checking and global reset (extended abstract). In Gerard Tel and Paul M. B. Vitányi, editors, *Distributed Algorithms, 8th International Workshop, WDAG '94, Terschelling, The Netherlands, September 29 - October 1, 1994, Proceedings*, volume 857 of *Lecture Notes in Computer Science*, pages 326–339. Springer, 1994.

[8] Baruch Awerbuch and George Varghese. Distributed program checking: a paradigm for building self-stabilizing distributed protocols (extended abstract). In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 258–267. IEEE Computer Society, 1991.

[9] Mor Baruch, Pierre Fraigniaud, and Boaz Patt-Shamir. Randomized proof-labeling schemes. In Chryssis Georgiou and Paul G. Spirakis, editors, *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 315–324. ACM, 2015.

[10] Catriel Beeri, Paris C. Kanellakis, François Bancilhon, and Raghu Ramakrishnan. Bounds on the propagation of selection into logic programs. *J. Comput. Syst. Sci.*, 41(2):157–180, 1990.

[11] Lélia Blin, Pierre Fraigniaud, and Boaz Patt-Shamir. On proof-labeling schemes versus silent self-stabilizing algorithms. In Pascal Felber and Vijay K. Garg, editors, *Stabilization, Safety, and Security of Distributed Systems - 16th International Symposium, SSS 2014, Paderborn, Germany, September 28 - October 1, 2014. Proceedings*, volume 8756 of *Lecture Notes in Computer Science*, pages 18–32. Springer, 2014.

[12] Sebastian Brandt, Klaus-Tycho Foerster, and Roger Wattenhofer. Augmenting flows for the consistent migration of multi-commodity single-destination flows in SDNs. *Pervasive and Mobile Computing*, pages –, 2016.

[13] Sebastian Brandt, Klaus-Tycho Förster, and Roger Wattenhofer. On consistent migration of flows in sdns. In *35th Annual IEEE International Conference on Computer Communications, INFOCOM 2016, San Francisco, CA, USA, April 10-14, 2016*, pages 1–9. IEEE, 2016.

[14] Martín Casado, Nate Foster, and Arjun Guha. Abstractions for software-defined networks. *Commun. ACM*, 57(10):86–95, 2014.

[15] Pavol Cerný, Nate Foster, Nilesh Jagnik, and Jedidiah McClurg. Optimal consistent network updates in polynomial time. In Cyril Gavoille and David Ilcinkas, editors, *Distributed Computing - 30th International Symposium, DISC 2016, Paris, France, September 27-29, 2016. Proceedings*, volume 9888 of *Lecture Notes in Computer Science*, pages 114–128. Springer, 2016.

[16] Shlomi Dolev. *Self-Stabilization.* MIT Press, 2000.

[17] Yuval Emek, Christoph Pfister, Jochen Seidel, and Roger Wattenhofer. Anonymous networks: randomization = 2-hop coloring. In Magnús M. Halldórsson and Shlomi Dolev, editors, *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*, pages 96–105. ACM, 2014.

[18] Klaus-Tycho Foerster, Thomas Luedi, Jochen Seidel, and Roger Wattenhofer. Local checkability, no strings attached. In *Proceedings of the 17th International Conference on Distributed Computing and Networking*, ICDCN '16, pages 21:1–21:10, New York, NY, USA, 2016. ACM.

[19] Klaus-Tycho Foerster, Ratul Mahajan, and Roger Wattenhofer. Consistent updates in software defined networks: On dependencies, loop freedom, and blackholes. In *2016 IFIP Networking Conference, Networking 2016 and Workshops, Vienna, Austria, May 17-19, 2016*, pages 1–9. IEEE, 2016.

[20] Klaus-Tycho Foerster, Stefan Schmid, and Stefano Vissicchio. Survey of consistent network updates. *CoRR*, abs/1609.02305, 2016.

[21] Klaus-Tycho Foerster and Roger Wattenhofer. The power of two in consistent network updates: Hard loop freedom, easy flow migration. In *25th International Conference on Computer Communication and Networks, ICCN 2016, Waikoloa, HI, USA, August 1-4, 2016*, pages 1–9. IEEE, 2016.

[22] Pierre Fraigniaud, Mika Göös, Amos Korman, Merav Parter, and David Peleg. Randomized distributed decision. *Distributed Computing*, 27(6):419–434, 2014.

[23] Pierre Fraigniaud, Mika Göös, Amos Korman, and Jukka Suomela. What can be decided locally without identifiers? In Panagiota Fatourou and Gadi Taubenfeld, editors, *ACM Symposium on Principles of Distributed Computing, PODC '13, Montreal, QC, Canada, July 22-24, 2013*, pages 157–165. ACM, 2013.

[24] Pierre Fraigniaud, Magnús M. Halldórsson, and Amos Korman. On the impact of identifiers on local decision. In Roberto Baldoni, Paola Flocchini, and Binoy Ravindran, editors, *Principles of Distributed Systems, 16th International Conference, OPODIS 2012, Rome, Italy, December 18-20, 2012. Proceedings*, volume 7702 of *Lecture Notes in Computer Science*, pages 224–238. Springer, 2012.

[25] Pierre Fraigniaud, Juho Hirvonen, and Jukka Suomela. Node labels in local decision. In Christian Scheideler, editor, *Structural Information and Communication Complexity - 22nd International Colloquium, SIROCCO 2015, Montserrat, Spain, July 14-16, 2015, Post-Proceedings*, volume 9439 of *Lecture Notes in Computer Science*, pages 31–45. Springer, 2015.

[26] Pierre Fraigniaud, Amos Korman, and David Peleg. Towards a complexity theory for local distributed computing. *J. ACM*, 60(5):35, 2013.

[27] Pierre François and Olivier Bonaventure. Avoiding transient loops during the convergence of link-state routing protocols. *IEEE/ACM Trans. Netw.*, 15(6):1280–1292, 2007.

[28] Pierre François, Clarence Filsfils, John Evans, and Olivier Bonaventure. Achieving sub-second IGP convergence in large IP networks. *Computer Communication Review*, 35(3):35–44, 2005.

[29] Mika Göös and Jukka Suomela. Locally checkable proofs. In Cyril Gavoille and Pierre Fraigniaud, editors, *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, PODC 2011, San Jose, CA, USA, June 6-8, 2011*, pages 159–168. ACM, 2011.

[30] Keqiang He, Junaid Khalid, Aaron Gember-Jacobson, Sourav Das, Chaithan Prakash, Aditya Akella, Li Erran Li, and Marina Thottan. Measuring control plane latency in sdn-enabled switches. In Jennifer Rexford and Amin Vahdat, editors, *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research, SOSR '15, Santa Clara, California, USA, June 17-18, 2015*, pages 25:1–25:6. ACM, 2015.

[31] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. Achieving high utilization with software-driven WAN. In Dah Ming Chiu, Jia Wang, Paul Barford, and Srinivasan Seshan, editors, *ACM SIGCOMM 2013 Conference, SIGCOMM'13, Hong Kong, China, August 12-16, 2013*, pages 15–26. ACM, 2013.

[32] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. B4: experience with a globally-deployed software defined wan. In Dah Ming Chiu, Jia Wang, Paul Barford, and Srinivasan Seshan, editors, *ACM SIGCOMM 2013 Conference, SIGCOMM'13, Hong Kong, China, August 12-16, 2013*, pages 3–14. ACM, 2013.

[33] Xin Jin, Hongqiang Harry Liu, Rohan Gandhi, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Jennifer Rexford, and Roger Wattenhofer. Dynamic scheduling of network updates. In Fabián E. Bustamante, Y. Charlie Hu, Arvind Krishnamurthy, and Sylvia Ratnasamy, editors, *ACM SIGCOMM 2014 Conference, SIGCOMM'14, Chicago, IL, USA, August 17-22, 2014*, pages 539–550. ACM, 2014.

[34] Amos Korman and Shay Kutten. Distributed verification of minimum spanning trees. *Distributed Computing*, 20(4):253–266, 2007.

[35] Amos Korman, Shay Kutten, and David Peleg. Proof labeling schemes. *Distributed Computing*, 22(4):215–233, 2010.

[36] Maciej Kuzniar, Peter Peresíni, and Dejan Kostic. What you need to know about SDN flow tables. In Jelena Mirkovic and Yong Liu, editors, *Passive and Active Measurement - 16th International Conference, PAM 2015, New York, NY, USA, March 19-20, 2015, Proceedings*, volume 8995 of *Lecture Notes in Computer Science*, pages 347–359. Springer, 2015.

[37] Arne Ludwig, Szymon Dudycz, Matthias Rost, and Stefan Schmid. Transiently secure network updates. In *Proc. ACM SIGMETRICS*, 2016.

[38] Arne Ludwig, Jan Marcinkowski, and Stefan Schmid. Scheduling loop-free network updates: It's good to relax! In Chryssis Georgiou and Paul G. Spirakis, editors, *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 13–22. ACM, 2015.

[39] Arne Ludwig, Matthias Rost, Damien Foucard, and Stefan Schmid. Good network updates for bad packets: Waypoint enforcement beyond destination-based routing policies. In Ethan Katz-Bassett, John S. Heidemann, Brighten Godfrey, and Anja Feldmann, editors, *Proceedings of the 13th ACM Workshop on Hot Topics in Networks, HotNets-XIII, Los Angeles, CA, USA, October 27-28, 2014*, pages 15:1–15:7. ACM, 2014.

[40] Ratul Mahajan and Roger Wattenhofer. On consistent updates in software defined networks. In Dave Levine, Sachin Katti, and Dave Oran, editors, *Twelfth ACM Workshop on Hot Topics in Networks, HotNets-XII, College Park, MD, USA, November 21-22, 2013*, pages 20:1–20:7. ACM, 2013.

[41] Moni Naor and Larry J. Stockmeyer. What can be computed locally? In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 184–193. ACM, 1993.

[42] Mark Reitblatt, Nate Foster, Jennifer Rexford, Cole Schlesinger, and David Walker. Abstractions for network update. In Lars Eggert, Jörg Ott, Venkata N. Padmanabhan, and George Varghese, editors, *ACM SIGCOMM 2012 Conference, SIGCOMM '12, Helsinki, Finland - August 13 - 17, 2012*, pages 323–334. ACM, 2012.

[43] Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. *SIAM J. Comput.*, 41(5):1235–1265, 2012.

[44] Stefan Schmid and Jukka Suomela. Exploiting locality in distributed SDN control. In Nate Foster and Rob Sherwood, editors, *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN 2013, The Chinese University of Hong Kong, Hong Kong, China, Friday, August 16, 2013*, pages 121–126. ACM, 2013.

[45] Nick Shelly, Brendan Tschaen, Klaus-Tycho Förster, Michael Alan Chang, Theophilus Benson, and Laurent Vanbever. Destroying networks for fun (and profit). In Jaudelice de Oliveira, Jonathan Smith, Katerina J. Argyraki, and Philip Levis, editors, *Proceedings of the 14th ACM Workshop on Hot Topics in Networks, Philadelphia, PA, USA, November 16 - 17, 2015*, pages 6:1–6:7. ACM, 2015.

[46] Stefano Vissicchio and Luca Cittadini. Flip the (flow) table: Fast lightweight policy-preserving sdn updates. In *2016 IEEE Conference on Computer Communications, INFOCOM 2016, San Francisco, CA, USA, April 10-15, 2016*. IEEE, 2016.