Certification of Compact Low-Stretch Routing Schemes*

Alkida Balliu[†1] **and Pierre Fraigniaud**[‡2]

1 **Gran Sasso Science Institute, L'Aquila, Italy**
2 **CNRS and University Paris Diderot, France**

Abstract

On the one hand, the correctness of routing protocols in networks is an issue of utmost importance for guaranteeing the delivery of messages from any source to any target. On the other hand, a large collection of *routing schemes* have been proposed during the last two decades, with the objective of transmitting messages along short routes, while keeping the routing tables small. Regrettably, all these schemes share the property that an adversary may modify the content of the routing tables with the objective of, e.g., blocking the delivery of messages between some pairs of nodes, without being detected by any node.

In this paper, we present a simple *certification* mechanism which enables the nodes to locally detect any alteration of their routing tables. In particular, we show how to locally verify the stretch-3 routing scheme by Thorup and Zwick [SPAA 2001] by adding certificates of $\widetilde{O}(\sqrt{n})$ bits at each node in n-node networks, that is, by keeping the memory size of the same order of magnitude as the original routing tables. We also propose a new *name-independent* routing scheme using routing tables of size $\widetilde{O}(\sqrt{n})$ bits. This new routing scheme can be locally verified using certificates on $\widetilde{O}(\sqrt{n})$ bits. Its stretch is 3 if using handshaking, and 5 otherwise.

1998 ACM Subject Classification C.2.2 Network Protocols, C.2.4 Distributed Systems, G.2.2 Graph Theory

Keywords and phrases Distributed verification, compact routing, local computing

Digital Object Identifier 10.4230/LIPIcs.DISC.2017.6

1 Introduction

Context. A *routing scheme* is a mechanism enabling to deliver messages from any source to any target in a network. The latter is typically modeled as an undirected connected weighted graph $G = (V, E)$ where V models the set of routers and E models the set of communication links between routers. All edges incident to a degree-d node are labeled from 1 to d, in an arbitrary manner, and the label at a node u of an incident edge e is called the *port number* of edge e at u. A routing scheme consists of a way of assigning a routing *table* to every node of the given network. These tables should contain enough information so that, for every target node t, each node is able to compute the port number of the incident edge through which it should forward a message of destination t. The routing tables must collectively guarantee that every message of any source s and any target t will eventually be delivered to t.

* A full version of the paper is available at `https://arxiv.org/abs/1704.06070`.
† Additional support from ANR Project DESCARTES. This work was done during the first author visit to Institut de Recherche en Informatique Fondamentale (IRIF) in 2016-2017.
‡ Additional support from ANR Project DESCARTES and Inria Project GANG.

Two scenarios are generally considered in the literature. One scenario allows the routing scheme to assign *names* to the nodes, and each target is then identified by its given name. The other, called *name independent*, is assuming that fixed names are given a priori (typically, a name is restricted to be the identity of a node), and the scheme cannot take benefit of naming nodes for facilitating routing.

Among many criteria for evaluating the quality of routing schemes, including, e.g., the time complexity for constructing the routing tables, the two main parameters characterizing a routing scheme are the *size* of its routing tables and names, and the *stretch*. The stretch of a routing scheme is the maximum, taken over all pairs of source-target nodes, of the ratio between the length of the route generated by the scheme from the source to the target, and the length of a shortest path between these two nodes. During the last two decades, there has been an enormous effort to design *compact* routing scheme (i.e., schemes using small tables) of *low* stretch (i.e., with stretch upper bounded by a constant) – see, e.g., [1, 2, 3, 4, 11, 16, 21, 22, 24]. A breakthrough was achieved in [24] where almost tight tradeoffs between size and stretch were explicitly demonstrated. In particular, [24] showed how to design a routing scheme with tables of size $\widetilde{O}(\sqrt{n})$ bits and stretch 3, in any network[1].

All the aforementioned routing schemes share the property that nodes do not have the capability to realize that the routing tables have been modified (either involuntarily or by an attacker). That is, a group of nodes may be provided with routing tables which look consistent with a desired routing scheme, but which do not achieve the desired performances of that scheme (e.g., large stretch, presence of loops, etc.). Indeed, the nodes are not provided with sufficient information to detect such an issue locally, that is, by having each node inspecting only the network structure and the tables assigned to nodes in its vicinity.

Objective. The objective of this paper is, given a routing scheme, to design a mechanism enabling each node to locally detect the presence of falsified routing tables, in the following sense. If some tables are erroneous, then at least one node must be able to detect that error by running a verification algorithm exchanging messages only between neighboring nodes.

Our mechanism for locally verifying the correctness of routing tables is inspired from proof-labeling schemes [20]. It is indeed based on assigning to each node a *certificate*, together with its routing table, and designing a distributed verification algorithm that checks the consistency of these certificates and tables by having each node inspecting only its certificate and its routing table, and the certificate and routing table of each of its neighbors. The set of certificates assigned to the nodes and the verification algorithm running at all nodes in parallel must satisfy that: (1) if all tables are correctly set, then, with some appropriate certificates, all nodes *accept*, and (2) if one or more tables are incorrectly set, then, for every assignment of the certificates, at least one node must *reject*. The second condition guarantees that the verification algorithm cannot be cheated: if the tables are incorrect, there are no ways of assigning the certificates such that all nodes accept.

Rephrasing the objective of the paper, our goal is to assign certificates to nodes, of size not exceeding the size of the routing tables, enabling the nodes to collectively verify the correctness of the routing tables, by having each node interacting with its neighbors only.

Our Results. We show how to locally verify the stretch-3 size-$\widetilde{O}(\sqrt{n})$ routing scheme by Thorup and Zwick [24]. Our certification mechanism uses certificates of $\widetilde{O}(\sqrt{n})$ bits at each node, that is, these certificates have size of the same order of magnitude as the original routing tables. Hence, verifying the scheme in [24] can be done without modifying the scheme,

[1] The notations $\widetilde{O}$ and $\widetilde{\Omega}$ ignore polylogarithmic factors.

Table 1 Summary of our results compared to previous work. All the listed routing schemes have space complexity of $\widetilde{O}(\sqrt{n})$ bits. Our verification algorithms use certificates on $\widetilde{O}(\sqrt{n})$ bits.

scheme	stretch	name independent	verifiable	comment
[24]	3	no	**yes**	–
[3]	5	yes	**?**	–
[2]	3	yes	**?**	–
this paper	**5**	**yes**	**yes**	–
this paper	**3**	**yes**	**yes**	handshaking

and without increasing the memory space consumed by that scheme. We also show that the same holds for the whole hierarchy of routing schemes proposed in [24] for providing a tradeoff between size and stretch.

The situation appears to be radically different for name-independent routing schemes. The stretch-3 name-independent routing scheme by Abraham et al. [2] also uses tables of size $\widetilde{O}(\sqrt{n})$ bits. However, each table includes references to far away nodes, whose validity does not appear to be locally verifiable using certificates of reasonable size. On the other hand, a simplified version of the scheme in [2] can be verified locally with certificates of size $\widetilde{O}(\sqrt{n})$ bits, but its stretch becomes at least 7. Therefore, we propose a new name-independent routing scheme, with tables of size $\widetilde{O}(\sqrt{n})$ bits that can be verified using certificates on $\widetilde{O}(\sqrt{n})$ bits as well. This new routing scheme has stretch at most 5, and the stretch can even be reduced to 3 using handshaking[2]. The routing scheme of Arias et al. [3] has also stretch 5, but it does not appear to be locally verifiable with certificates of reasonable size, and using handshaking does not enable to reduce the stretch.

All our results are summarized in Table 1.

Related Work. The design of *compact* routing tables, and the explicit identification of tradeoffs between the table size and the routes length was initiated thirty years ago, with the seminal work in [22] and [21]. Since then, a large amount of papers were published on this topic, aiming at refining these tradeoffs, and at improving different aspects of the routing schemes, including routing in specific classes of graphs (see [15, 17]). In particular, routing schemes were designed for trees in [11, 24], with space complexity $O(\log^2 n/\log\log n)$ bits[3]. This space complexity was shown to be optimal in [12].

It was proved [18] that, in n-node networks, any shortest path routing scheme requires tables of size $\widetilde{\Omega}(n)$ bits. The aforementioned routing scheme in [24] with stretch 3 and space complexity $\widetilde{O}(\sqrt{n})$ bits was shown to be optimal in [16], in the following sense: no routing scheme with space complexity $o(n)$ bits can achieve a stretch $s < 3$, and, assuming the correctness of a conjecture by Erdős regarding a tradeoff between girth and edge density in graphs, every routing scheme with stretch $s < 5$ has space complexity $\Omega(\sqrt{n})$ bits. On the positive side, [24] tightens the size-stretch tradeoff of [21] by showing that, for every $k \geq 2$, there exists a routing scheme with stretch $4k-5$ and space complexity $\widetilde{O}(n^{1/k})$ bits. (The stretch can be reduced to $2k-1$ using handshaking). Recently, [8] showed that, for $k \geq 4$, a stretch $s = \alpha\, k$ with $\alpha < 4$ can be achieved using routing tables of size $\widetilde{O}(n^{1/k})$.

[2] The handshaking mechanism is similar to DNS lookup in TCP/IP. It allows querying some node(s) for getting additional information about the route to the target.

[3] The space complexity can be reduced to $O(\log n)$ if the designer of the routing scheme is also allowed to assign the port numbers to the nodes.

The distinction between name-independent routing schemes, and routing schemes assigning specific names to the nodes was first made in [4]. Then, [5] presented techniques for designing name-independent routing schemes with constant stretch and space complexity $o(n)$ bits. Almost 15 years after, [3] described a name-independent routing scheme with stretch 5 and space complexity $\widetilde{O}(\sqrt{n})$ bits. This was further improved in [2] thanks to a name-independent routing scheme with stretch 3 and space complexity $\widetilde{O}(\sqrt{n})$ bits. A couple of years later, [1] showed that there are tradeoffs between stretch and space complexity for name-independent routing schemes as well. Specifically, [1] showed that, for any $k \geq 1$, there exists a name-independent routing scheme with space complexity $\widetilde{O}(n^{1/k})$ bits and stretch $O(k)$.

The certification mechanism used in this paper is based on the notion of proof-labeling scheme introduced in [20] in which an oracle, called *prover*, assigns certificates to the nodes, and a distributed algorithm, called *verifier*, checks that this certificates collectively form a proof that the global state of the network is legal with respect to a given boolean network predicate. Proof-labeling schemes have been widely used in literature. For example, [23] uses them to verify spanning trees in networks. This result has been extended in [14], where proof-labeling schemes are used to verify spanning trees in evolving networks that are evolving with time. Variants of proof-labeling schemes have been considered in, e.g., [7, 13], and [19]. More generally, see [10] for a survey of distributed decision.

2 Definitions

Routing Schemes. Let $\mathcal{F}$ be a family of edge-weighted graphs with edges labeled at each node by distinct port numbers from 1 to the degree of the node. The weights are all positive, and the weight of edge e represents its length. It is thus denoted by $\mathrm{length}(e)$. The nodes are given distinct identities. All node-identities and edge-weights are supposed to be stored on $O(\log n)$ bits. For the sake of simplifying notations, we do not make a distinction between a node v and its identity, also denoted by v. Given two nodes u, v, we denote by $\delta(u, v)$ the weighted distance between u and v. Given an edge e of end-point u, the port number of e at u is denoted by $\mathsf{port}_u(e)$.

We follow the usual setting of proof-labeling scheme [20] in which the values of the edge-weights, node identities, and port numbers are fixed, and cannot be corrupted. Only the internal memories of the nodes, storing information about, say, routing, are susceptible to be corrupted.

A *routing scheme* for $\mathcal{F}$ is a mechanism assigning a *name*, $\mathsf{name}(u)$, and a routing *table*, $\mathsf{table}(u)$, to every node u of every graph $G \in \mathcal{F}$ such that, for any pair (s, t) of nodes of any $G \in \mathcal{F}$, there exists a path $u_0, u_1, \ldots, u_k$ from s to t in G with $u_0 = s$, $u_k = t$, and

$$\mathsf{table}(u_i)\big(\mathsf{name}(t)\big) = \mathsf{port}_{u_i}(\{u_i, u_{i+1}\}) \tag{1}$$

for every $i = 0, \ldots, k-1$. That is, every intermediate node u_i, $0 \leq i < k$, can determine on which of its ports the message has to be forwarded, based solely on its routing table, and on the name of the target. In Eq. 1, each table is viewed as a function taking names as arguments, and returning port numbers. The path $u_0, u_1, \ldots, u_k$ is then called the *route* generated by the scheme from s to t. It is worth pointing out the following observations.

Name-independent routing schemes are restricted to use names that are fixed a priori, that is, the name of a node is its identity, i.e., $\mathsf{name}(v) = v$ for every node v. Instead, name-dependent routing schemes allow names to be set for facilitating routing, and names are typically just bounded to be storable on a polylogarithmic number of bits.

The *header* of a message is the part of that message containing all information enabling its routing throughout the network. The header of a message with destination t is typically $\mathsf{name}(t)$. However, some routing schemes ask for message headers that can be modified. This holds for both name-dependent schemes (like, e.g., [24]) and name-independent schemes (like, e.g., [2]). The typical scenario requiring modifiable headers is when a message is routed from source s to target t as follows. From $\mathsf{name}(t)$ and $\mathsf{table}(s)$, node s can derive the existence of some node v containing additional information about how to reach t. Then the message is first routed from s to v, and then from v to t. Distinguishing these two distinct parts of the routes from s to t often requires to use different headers. In case of modifiable headers, Eq. (1) should be tuned accordingly as the argument of routing is not necessarily just a name, but a header.

Some routing schemes may use a mechanism called *handshaking* [24], which is an abstraction of mechanisms such as Domain Name System (DNS) enabling to recover an IP address from its domain name. Let us consider the aforementioned scenario where a routing scheme routes a message from s to t via an intermediate node v identified by s from $\mathsf{name}(t)$ and $\mathsf{table}(s)$. One can then enhance the routing scheme by a handshaking mechanism, enabling s to query v directly, and to recover the information stored at v about t. Then s can route the message directly from s to t, avoiding the detour to v. That is, handshaking enables to distinguish the part of the routing used to get information about the target (like in DNS), from the part of routing used to transfer messages (like in IP). Handshaking is used in [24] to reduce the stretch of routing schemes with space complexity $\widetilde{O}(n^{1/k})$ bits from $4k-5$ to $2k-1$, for every $k>2$.

The *size* of a routing scheme is the maximum, taken over all nodes u of all graphs in $\mathcal{F}$, of the memory space required to encode the function $\mathsf{table}(u)$ at node u. The *stretch* of a routing scheme is the maximum, taken over all pairs (s,t) of nodes in all graphs $G\in\mathcal{F}$, of the ratio of the length of the route from s to t (i.e., the sum of the edge weights along that route) with the weighted distance between s and t in G.

Distributed Verification. Given a graph G, a *certificate function* for G is a function $\mathsf{certificate}: V(G)\to\{0,1\}^*$ assigning a certificate, $\mathsf{certificate}(u)$, to every node $u\in V(G)$. A *verification* algorithm is a distributed algorithm running concurrently at all nodes in parallel. At every node $u\in V(G)$ of every graph $G\in\mathcal{F}$, the algorithm takes as input the identity of node u, the certificate $\mathsf{certificate}(u)$ and routing table $\mathsf{table}(u)$ assigned to node u, as well as the collection of pairs $(\mathsf{table}(v),\mathsf{certificate}(v))$ assigned to all neighbors v of u, with their identities, and outputs *accept* or *reject*.

▶ **Definition 1.** A routing scheme for $\mathcal{F}$ is *verifiable* if there exists a verification algorithm VERIF such that, for every $G\in\mathcal{F}$,

- if the tables given to all nodes of G are the ones specified by the routing scheme, then there exists a certificate function for G such that the verification algorithm VERIF outputs *accept* at all nodes;
- if some tables given to some nodes of G differ from the ones specified by the routing scheme, then, for every certificate function for G, the verification algorithm VERIF outputs reject in at least one node.

The second bullet guarantees that if an adversary modifies some routing tables, or even just a single bit of a single table, then there are no ways it can also modify some, or even all certificates so that to force all nodes to accept: at least one node will detect the change. Of course, this node does not need to be the same for different modifications of the routing tables, or for different certificates.

▶ Remark. The above definition is the classical definition of proof-labeling scheme applied to verifying routing schemes. In particular, it may well be the case that a correct labeling scheme be rejected if the certificates have not been set appropriately, just like a spanning tree T will be rejected by a proof-labeling scheme for spanning trees if the certificates have been set for another spanning tree $T' \neq T$.

3 Name-dependent routing scheme

In this section, we show how to verify the stretch-3 routing scheme by Thorup and Zwick in [24]. This scheme uses tables of $\widetilde{O}(\sqrt{n})$ bits of memory at each node. We show the following:

▶ **Theorem 2.** *The stretch-3 routing scheme by Thorup and Zwick in [24] can be locally verified using certificates of size $\widetilde{O}(\sqrt{n})$ bits.*

Before proving the theorem, let us recall the structure of the routing scheme in [24]. It assigns names and tables to nodes of every $G = (V, E)$ as follows.

Landmarks, Bunch and Clusters. The routing scheme in [24] uses the notion of *landmarks* (a.k.a. *centers*). These landmarks form a subset $L \subseteq V$ of nodes. For $v \in V$, let l_v denote the landmark closest to v in G. For every $v \in V$, the *bunch* of v with respect to the set L is defined as follows: $\mathsf{bunch}(v) = \{u \in V : \delta(v,u) < \delta(l_v, v)\}$. The routing scheme in [24] also uses the notion of *cluster*. For every node $v \in V$, $\mathsf{cluster}(v) = \{u \in V : \delta(v,u) < \delta(l_u, u)\}$. As a consequence, for every $u, v \in V$, we have: $u \in \mathsf{cluster}(v) \iff v \in \mathsf{bunch}(u)$. Note that since, for every $v \in V$, and every $l \in L$, we have $l \notin \mathsf{bunch}(v)$, it follows that $\mathsf{cluster}(l) = \emptyset$ for every $l \in L$. By construction of the bunches and the clusters, it also holds that, for every $v \in V$, $\mathsf{cluster}(v) \cap L = \emptyset$. Also, clusters satisfy the following property.

▶ **Lemma 3** ([24])**.** *If $u \in \mathsf{cluster}(v)$ then, for every node w on a shortest path between u and v, we have $u \in \mathsf{cluster}(w)$.*

In [24], the landmarks are chosen by an algorithm that samples them uniformly at random in V until the following holds: for every node v, $|\mathsf{cluster}(v)| < 4\sqrt{n}$. It is proved that this algorithm returns, w.h.p., a set of landmarks of size at most $2\log(n)\sqrt{n}$.

Names and tables. For every two nodes v and t, let $\mathsf{next}(v,t)$ be the port number of an edge incident to v on a shortest path between v and t. Each node $t \in V$ is assigned a $3\lceil\log(n)\rceil$-bit name as follows: $\mathsf{name}(t) = (t, l_t, \mathsf{next}(l_t, t))$. Each node $v \in V$ then stores the following $\widetilde{O}(\sqrt{n})$-bit information in its routing table, $\mathsf{table}(v)$:

- the identities of all the landmarks $l \in L$;
- the identities of all nodes $t \in \mathsf{cluster}(v)$;
- the set $\{\mathsf{next}(v,t) : t \in L \cup \mathsf{cluster}(v)\}$.

Routing. Note that, by Lemma 3, any message from a node v to a node in $\mathsf{cluster}(v)$ reaches its target along a shortest path. The same holds for landmarks since every node is given information about how to reach every landmark. In general, let us assume that v wants to send a message to some node t with label $(t, l_t, \mathsf{next}(l_t, t))$ that is neither a landmark nor belongs to $\mathsf{cluster}(v)$. In this case, v extracts the landmark l_t nearest to t from t's name (note here the impact of allowing the scheme to assign specific names to nodes), and forwards the message through the port on a shortest path towards l_t using the information v has in its table. Upon reception of the message, l_t forwards the message towards t on a shortest

path using $\mathsf{next}(l_t, t)$ (this information can also be extracted from the name of t), to reach a node $z \in \mathsf{bunch}(t)$. At last, since $z \in \mathsf{bunch}(t)$, we have $t \in \mathsf{cluster}(z)$, which means that z can route to t via a shortest path using the information available in its table. By Lemma 3, this also holds for every node along a shortest path between z and t. Using symmetry, and triangle inequality, [24] shows that this routing scheme guarantees stretch 3.

Proof of Theorem 2. To enable local verification of the stretch-3 routing scheme in [24], a certificate of size $\widetilde{O}(\sqrt{n})$ bits is given to each node. Let $G = (V, E)$ be an undirected graph with positive weights assigned to its edges, and a correct assignment of the routing tables to the nodes according to the specifications of [24] as summarized before. Then each node $v \in V$ is assigned a certificate composed of:

- the distance between v and every landmark in L;
- the distance between v and every node in $\mathsf{cluster}(v)$;
- the set $\{\delta(t, l_t) : t \in \mathsf{cluster}(v)\}$.

As claimed, all these information can be stored using $\widetilde{O}(\sqrt{n})$ bits of memory.

We assume, without loss of generality, that all nodes know n (verifying the value of n is easy using a proof-labeling scheme with $O(\log n)$-bit certificates [20]). The verification of the routing scheme then proceeds as follows. We describe the verification algorithm VERIF running at node $v \in V$. This verification goes in a sequence of steps. At each step, either v outputs *reject* and stops, or it goes to the next step.

We denote by $L(v)$, $C(v)$, and $\{N(v, t) : t \in L(v) \cup C(v)\}$ the content of the routing table of v. These entries are supposed to be the set of landmarks, the cluster of v, and the set of next-pointers given to v, respectively. We also denote by d the distance given in the certificates. That is, node v is given a set $\{d(v, t) : t \in L(v) \cup C(v)\}$ and a set $\{d(t, l_t) : t \in C(v)\}$ where l_t is supposed to be the node in $L(v)$ closest to $t \in C(v)$. Of course, if a node does not have a table and a certificate of these forms, then it outputs *reject*. So, we assume that all tables and certificates have the above formats. The algorithm proceeds as follows at every node v. Node v checks that

1. the information in its table satisfy $|C(v)| \leq 4\sqrt{n}$ and $|L(v)| \leq 2 \log n \sqrt{n}$ bits;
2. it has the same set of landmarks as its neighbors;
3. for every $l \in L(v)$ with $l \neq v$, there exists a neighbor u of v satisfying $d(v, l) = \text{length}(\{v, u\}) + d(u, l)$, and all other neighbors satisfy $d(v, l) \leq \text{length}(\{v, u\}) + d(u, l)$, with $N(v, l)$ pointing to a neighbor u satisfying $d(v, l) = \text{length}(\{v, u\}) + d(u, l)$;
4. if $v \in L(v)$ then $C(v) = \emptyset$;
5. if $v \notin L(v)$ then $v \in C(v)$ and, for every node $t \in C(v)$ with $t \neq v$, there exists a neighbor u of v satisfying $t \in C(u)$ with $d(v, t) = \text{length}(\{v, u\}) + d(u, t)$, and every neighbor u of v with $t \in C(u)$ satisfies $d(v, t) \leq \text{length}(\{v, u\}) + d(u, t)$, with $N(v, t)$ pointing to a neighbor u satisfying $t \in C(u)$ and $d(v, t) = \text{length}(\{v, u\}) + d(u, t)$;
6. for every node $t \in C(v)$, for every neighbor u of v satisfying $t \in C(u)$, the distance $d(t, l_t)$ in the certificate of u is equal to the distance $d(t, l_t)$ in the certificate of v;
7. for every $t \in C(v)$, it holds that $d(v, t) < d(t, l_t)$;
8. for every neighbor u, and every $t \in \mathsf{cluster}(u) \setminus \mathsf{cluster}(v)$, it holds that $\text{length}\{(v, u)\} + d(u, t) \geq d(t, l_t)$.

If v passes all the above tests, then v outputs *accept*, else it outputs *reject*.

We now establish the correctness of this local verification algorithm. First, by construction, if all tables are set according to [24], that is, if, for every node v, $L(v) = L$, $C(v) = \mathsf{cluster}(v)$ and $N(v, t) = \mathsf{next}(v, t)$ for all $t \in L \cup \mathsf{cluster}(v)$, then every node running the verification

algorithm with the appropriate certificate $\{\delta(v,t) : t \in L \cup \mathsf{cluster}(v)\} \cup \{\delta(t,l_t) : t \in \mathsf{cluster}(v)\}$ where l_t is the node in L closest to $t \in \mathsf{cluster}(v)$, will face no inconsistencies with its neighbors, i.e., all the above tests are passed, leading every node to accept, as desired.

So, it remains to show that if some tables are not what they should be according to [24], then, no matter the certificates assigned to the nodes, at least one node will fail one of the tests. If all nodes output *accept*, then, by Step 1, all routing tables are of the appropriate size. Also, by Step 2, the set L of landmarks given to the nodes is the same for all nodes, as otherwise there will be two neighbors that would have different sets. Moreover, by Step 3, we get that, at every node v, the distances of this node to the landmarks, as stored in its certificate, are correct, from which we infer that $N(v,l)$ is appropriately set in the table of v, that is $N(v,l) = \mathsf{next}(v,l)$ for every $l \in L$. Hence, if all the tests in Steps 1-3 are passed, all the data referring to L in both the tables and the certificates are consistent. In particular, every node v knows the landmark l_v which is closest to it, and its distance $\delta(v,l_v)$.

We now show that if all these tests as well as the remaining tests are passed, then the clusters in the tables are correct, w.r.t. L, as well as the next-pointers. We first show that, if all tests are passed, then, for every node $v \in V$, $C(v) \subseteq \mathsf{cluster}(v)$. By Step 4, this latter equality holds for every landmark v. By Step 5, we get that, at every node v, the distance of this node to every node $t \in C(u)$, as stored in its certificate, are correct, from which we infer that $N(v,t)$ is appropriately set in the table of v, that is $N(v,t) = \mathsf{next}(v,t)$ for every $t \in C(v)$. By Step 6, we get that, for every $t \in C(v)$, we do have $d(l_t,t) = \delta(l_t,t)$. Indeed, this equality will be checked by all nodes on a shortest path between v and t (whose existence is guaranteed by Step 5), and t has the right distance $\delta(t,l_t)$ in its certificate by Step 3. Recall that $\mathsf{cluster}(v) = \{t \in V \mid \delta(v,t) < \delta(l_t,t)\}$ where l_t is the landmark closest to t. Step 7 precisely checks that inequality.

It remains to show that there are no nodes in $\mathsf{cluster}(v)$ that are not in $C(v)$. Assume that there exists $t \in \mathsf{cluster}(v) \setminus C(v)$, and let P be a shortest path between v and t. Let v' be the closest node to v on P such that $t \in C(v') \subseteq \mathsf{cluster}(v')$. Note that such a node v' exists as $t \in C(t)$. Let v'' be the node just before v' on P traversed from v to t. By Lemma 3, since $t \in \mathsf{cluster}(v)$, we also have $t \in \mathsf{cluster}(v'')$. We have $t \in \mathsf{cluster}(v'') \setminus C(v'')$. Therefore $\delta(v'',t) < \delta(l_t,t)$. Now, $\delta(v'',t) = \mathrm{length}(\{(v'',v')\}) + \delta(v',t)$ because P is a shortest path between v'' and t passing through v'. So, $\mathrm{length}(\{(v'',v')\}) + \delta(v',t) < \delta(l_t,t)$. Step 8 guarantees that it is not the case. Therefore, there are no nodes in $\mathsf{cluster}(v) \setminus C(v)$. It follows that $\mathsf{cluster}(v) = C(v)$ for all nodes v. This completes the proof of Theorem 2. ◀

4 Name-independent Routing Scheme

The purpose of this section is twofold. First, it serves as recalling basic notions that will be helpful for the design of our new name-independent routing scheme. Second, it is used to show why the known name-independent routing scheme in [2] appears to be difficult, and perhaps even impossible to verify locally.

The Routing Scheme of [2]

The stretch-3 name-independent routing scheme of [2] uses $\widetilde{O}(\sqrt{n})$ space at each node. We provide a high level description of that scheme. Recall that, in name-independent routing, a target node is referred only by its identity. That is, $\mathsf{name}(t)$ is the identity of t, i.e., $\mathsf{name}(t) = t$. Let $G = (V,E)$. For every node $v \in V$, the *vicinity ball* of v, denoted by $\mathsf{ball}(v)$, is the set of the $4\lceil \alpha \log(n) \sqrt{n} \rceil$ closest nodes to v, for a large enough constant $\alpha > 0$, where ties are broken using the order of node identities. By this definition, if $u \in \mathsf{ball}(v)$ and w is on a shortest path between v and u, then $u \in \mathsf{ball}(w)$.

Color-Sets. In [2], the nodes are partitioned into sets $C_1, \ldots C_{\sqrt{n}}$, called *color-sets*, and, for $i = 1, \ldots, \sqrt{n}$, the nodes in color-set C_i are assigned the same color i. For every node $v \in V$, its vicinity ball, $\mathsf{ball}(v)$, contains at least one node from each color-set. To get this, the color of a given node v is determined by a hash function *color* which, given the identity of a node, maps that identity to a color in $\{1, \ldots, \sqrt{n}\}$. This mapping from identities to colors is balanced in the sense that at most $O(\log n\sqrt{n})$ nodes map to the same color. A color is chosen arbitrarily, and all nodes with that color are considered to be the landmarks. Let L be the set of landmarks. It holds that $|L| = O(\log n\sqrt{n})$. Also, each node $v \in V$ has at least one landmark in its vicinity ball. We fix, for each vicinity ball $\mathsf{ball}(v)$, an arbitrary landmark, denoted by l_v.

Routing in trees. This construction in [2] makes use of routing schemes in trees. More precisely, they use the results from [11, 24], which states that there exists a shortest-path (name-dependent) routing scheme for trees using names and tables both on $O(\log^2(n)/\log\log(n))$ bits in n-node trees. For a tree T containing node v, let $\mathsf{table}_T(v)$ and $\mathsf{name}_T(v)$ denote the routing table of node v in T, and the name of v in T, as assigned by the scheme in [11].

The routing tables. For any node v, let $T(v)$ be a shortest-path spanning tree rooted at v. Let $P(v, w, u)$ be a path between v and u composed of a shortest path between v and w and a shortest path between w and u. Such a path is said to be *good* for (v, u) if $v \in \mathsf{ball}(w)$, and there exists an edge $\{x, y\}$ along a shortest path between w and u with $x \in \mathsf{ball}(w)$ and $y \in \mathsf{ball}(u)$; Every node $v \in V$ stores the following information in its routing table $\mathsf{table}(v)$:

- the hash function *color* that maps identities in colors;
- the identity of every node $u \in \mathsf{ball}(v)$, and the port number $\mathsf{next}(v, u)$;
- for every landmark $l \in L$, the routing table $\mathsf{table}_{T(l)}(v)$ for routing in $T(l)$;
- for every node $u \in \mathsf{ball}(v)$, the routing table $\mathsf{table}_{T(u)}(v)$ for routing in $T(u)$;
- the identities of all nodes with same color as v, and, for each such node u, the following additional information:

  Rule 1: if there are no good paths $P(v, w, u)$, v stores $\big(\mathsf{name}_{T(l_u)}(l_u), \mathsf{name}_{T(l_u)}(u)\big)$.

  Rule 2: if there exists a good path $P(v, w, u)$, then let us pick a good path P of minimum length among all good paths; then, let us compare its length $|P|$ with the length $|Q|$ of the path Q composed of a shortest path between v and l_u, and a shortest path between l_u and u; provide v with $\big(\mathsf{name}_{T(l_u)}(l_u), \mathsf{name}_{T(l_u)}(u)\big)$ if $|Q| \leq |P|$, and with $\big(\mathsf{name}_{T(v)}(w), x, \mathsf{port}_x(\{x, y\}), \mathsf{name}_{T(y)}(u)\big)$ otherwise.

Storing the hash function *color* requires $O(\sqrt{n})$ bits. L and $\mathsf{ball}(v)$ are both of size $\widetilde{O}(\sqrt{n})$ bits. Moreover, the number of nodes with identical color is $\widetilde{O}(\sqrt{n})$, for every color. Finally, shortest-path routing in any tree can be achieved using tables and names of size $O(\log^2 n/\log\log n)$ bits [11, 24]. It follows that $|\mathsf{table}(v)| = \widetilde{O}(\sqrt{n})$, as desired.

Routing. Routing from a source s to a target t is achieved in the following way (see [2] for more details). If $t \in \mathsf{ball}(s)$, or $t \in L$, or s and t have the same color, then s routes to t using the information available in its table. More specifically, if $t \in \mathsf{ball}(s)$ or $t \in L$, then s sets the header of the message as just the identity of the target t. Instead, if s and t have the same color but $t \notin \mathsf{ball}(s) \cup L$, the source s sets the header as one of the two possible rules 1 or 2. Otherwise, that is, if $t \notin \mathsf{ball}(s) \cup L$ and $\mathit{color}(s) \neq \mathit{color}(t)$, node s routes the message towards some node $w \in \mathsf{ball}(s)$ sharing the same color as t. (The color of t can be obtained by hashing the identity of t). The header is set to t, and w will change the header upon reception of this message, according to the rules previously specified. It is shown in [2] that this routing guarantees a stretch 3.

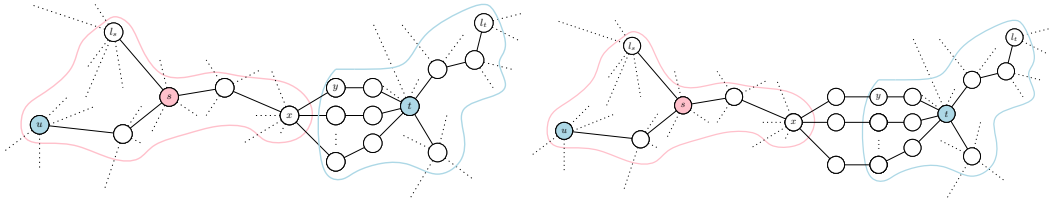

Figure 1 (Left) All the nodes in a shortest path between s and t belong to $\mathsf{ball}(s) \cup \mathsf{ball}(t)$. (Right) There are nodes in a shortest path between s and t that do not belong to $\mathsf{ball}(s) \cup \mathsf{ball}(t)$.

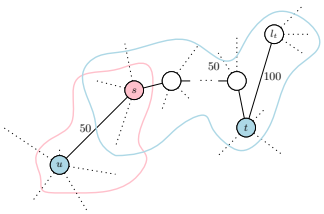

Figure 2 A route of stretch 7.

On the difficulty of locally verifying the scheme in [2]

We note that the routing scheme in [2], as sketched in the previous subsection, has some "*global*" features that makes it plausibly difficult to locally verify, by adding certificates of size $\widetilde{O}(\sqrt{n})$ bits at each node. In this subsection, we mention one of these global features, illustrated in the example depicted on Fig. 1. We are considering routing from a source $s \notin L$ to a target $t \notin L$, of different colors, with $s \notin \mathsf{ball}(t)$ and $t \notin \mathsf{ball}(s)$. Let us assume that node s has color red, while node t has color blue. According to the routing scheme in [2], node s first routes the message towards some blue node $u \in \mathsf{ball}(s)$ to get information about the blue target t. In the example of Fig. 1(left), node u stores $\big(\mathsf{name}_{T(u)}(s), x, port(x,y), \mathsf{name}_{T(y)}(t)\big)$, to guarantee a stretch 3. Instead, in the example of Fig. 1(right), node u stores $\big(\mathsf{name}_{T(l_t)}(l_t), \mathsf{name}_{T(l_t)}(t)\big)$. to guarantee such a small stretch. Verifying locally whether there exists a good path between s and t, which is the condition leading to distinguishing the case where the content of rule 1 must apply, from the case where the content of rule 2 must apply, appears to be a very difficult matter when restricted to certificates of size $\widetilde{O}(\sqrt{n})$.

We point out that systematically applying rule 1 would result in a routing scheme that may be locally verifiable. However, its stretch is at least 7. To see why, let us consider the example displayed in Figure 2 where $s \in \mathsf{ball}(t)$ but $t \notin \mathsf{ball}(s)$. The radius of $\mathsf{ball}(s)$ is 50. Let $u \in \mathsf{ball}(s)$ be one of the farthest nodes to s, i.e., $\delta(s,u) = 50$. The radius of $\mathsf{ball}(t)$ is 100 and $\delta(t, l_t) = 100$. Although $\delta(u,t) = 100$, we assume $u \notin \mathsf{ball}(t)$ due to lexicographical order priorities. Finally, assume that $\delta(s,t) = 50$. The worst case route from s to t would then be $s \rightsquigarrow u \rightsquigarrow s \rightsquigarrow t \rightsquigarrow l_t \rightsquigarrow t$. This path is of length $\delta(s,u) + \delta(u,s) + \delta(s,t) + \delta(t,l_t) + \delta(l_t,t) = 2 \times 50 + 50 + 2 \times 100 = 350$ which is $7\delta(s,t)$.

5 A New Name-Independent Routing Scheme

In this section, we describe and analyze a new name-independent routing scheme, denoted by $\mathcal{R}$. The section is entirely dedicated to the proof of the following theorem. Recall that, given a family of events $(\mathcal{E}_n)_{n\geq 1}$, event $\mathcal{E}_n$ holds with high probability if $\Pr[\mathcal{E}_n] = 1 - O(1/n^c)$ for some $c \geq 1$.

▶ **Theorem 4.** *The name-independent routing scheme $\mathcal{R}$ uses routing tables of size $\widetilde{O}(\sqrt{n})$ bits at each node. It guarantees stretch 5, and, using handshaking, its stretch can be reduced to 3. In both cases, $\mathcal{R}$ can be locally verified using certificates of size $\widetilde{O}(\sqrt{n})$ bits, using a 1-sided error verification scheme which guarantees that incorrect tables are detected with high probability.*

Our verifier used to establish Theorem 4 is actually deterministic. The certificates however store hash functions chosen at random. We assume that these hash functions are not corruptible, and that the adversary is not capable to create collisions (if not accidentally, by chance). Even though this assumption may seem strong, we point out that this is relevant to practical situations in which cryptographic hash functions designed to be collision resistant and hard to invert are used (for more details please refere to Chapter 5 of [9]).

The new routing scheme $\mathcal{R}$. Our new routing scheme $\mathcal{R}$ borrows ingredients from both [2] and [24]. In particular, the landmarks are chosen as for the (name-dependent) routing scheme in [24], i.e., in such a way that every node v has a cluster $\mathsf{cluster}(v)$ of size at most $4\sqrt{n}$, and the landmarks form a set L of size at most $2\log n\sqrt{n}$. However, we reinforce the way the nearest landmark to v is selected, by picking the nearest landmark l_v such that $l_v = \text{argmin}_{l\in L}\delta(v,l)$ where ties are broken by choosing the landmark with smallest identity. Also, we slightly reinforce the definition of next: For every two nodes v and t, we set $\mathsf{next}(v,t)$ as the *smallest* port number at v of an edge incident to v on a shortest path between v and t. These two reinforcements of the definitions of landmarks and next guarantee the following.

▶ **Lemma 5.** *Let $v \in V$, and let l_v be its landmark. Let u be a node on a shortest path between v and l_v. Then $l_u = l_v$.*

▶ **Lemma 6.** *Let $v \in V$, let l_v be its landmark, and let w be the neighbor of l_v such that $\mathsf{next}(l_v, v) = \mathsf{port}_{l_v}(\{l_v, w)\}$. Let u be a node on a shortest path between v and w. We have $\mathsf{next}(l_v, u) = \mathsf{next}(l_v, v)$.*

As in [2], each node that is not a landmark is given a color in $\{1,\ldots,\sqrt{n}\}$ determined by a hash function, *color*, that maps identities to colors, where at most $O(\log n\sqrt{n})$ nodes map to the same color. Also, each node v has a vicinity ball, $\mathsf{ball}(v)$, that contains the $O(\log(n)\sqrt{n})$ nodes closest to v (breaking ties using identities). This guarantees that, with high probability, for every node v, there is at least one node of each color in $\mathsf{ball}(v)$.

For each color c, where $1 \le c \le \sqrt{n}$, we define $\mathsf{Dir}_c = \{(v, l_v, \mathsf{next}(l_v, v)) : color(v) = c\}$ which includes the direction to take at l_v for reaching v of color c along a shortest path.

The routing tables. Every node $v \in V$ then stores the following information in $\mathsf{table}(v)$:

- the hash function *color* that maps identities in colors;
- the identity of every landmark $l \in L$, and the corresponding port $\mathsf{next}(v, l)$;
- the identity of every node $u \in \mathsf{cluster}(v)$, and the corresponding port $\mathsf{next}(v, u)$;
- the identity of every node $u \in \mathsf{ball}(v)$, and the corresponding port $\mathsf{next}(v, u)$;
- the set $\mathsf{Dir}_{\mathrm{color}(v)}$.

Storing the hash function *color* requires $O(\sqrt{n})$ bits, as we use the same function as in [2]. Since L, $\mathsf{cluster}(v)$, $\mathsf{ball}(v)$, and $\mathsf{Dir}_{\mathrm{color}(v)}$ are all of size $\widetilde{O}(\sqrt{n})$, we get that $|\mathsf{table}(v)| = \widetilde{O}(\sqrt{n})$ as desired.

Routing. Let us consider routing towards a target node t, and let v be the current node. If $t \in \mathsf{cluster}(v)$, then routing to t is achieved using $\mathsf{next}(v,t)$. Notice that, by Lemma 3, routing to t will actually be achieved along a shortest path. Similarly, if $t \in \mathsf{ball}(v)$, then routing to t is achieved using $\mathsf{next}(v,t)$ along a shortest path, and this also holds if t is a landmark. In general, i.e., if t is neither a landmark nor a node of $\mathsf{cluster}(v) \cup \mathsf{ball}(v)$, then node v computes $\mathit{color}(t)$ by hashing the identity of t.

If $\mathit{color}(t) = \mathit{color}(v)$, then v forwards the message towards l_t using the information in $\mathsf{Dir}_{\mathrm{color}(v)}$, and including $(l_t, \mathsf{next}(l_t,t))$ in the header of the message so that intermediate nodes carry on routing this message to l_t. At l_t, the message will be routed to t using the information $\mathsf{next}(l_t,t)$ available in the header, reaching a node u_t such that $t \in \mathsf{cluster}(u_t)$. At this point, routing proceeds to t along a shortest path.

If $\mathit{color}(t) \neq \mathit{color}(v)$, then the message is forwarded to an arbitrary node $w \in \mathsf{ball}(v)$ having the same color as t (we know that such a node exists), with w in the header of the message. The message then reach w along a shortest path. At w, we have $\mathit{color}(t) = \mathit{color}(w)$, and thus routing proceeds as in the previous case.

Handshaking. The routing with handshaking to node t proceeds as follows. If $t \in L \cup \mathsf{cluster}(v) \cup \mathsf{ball}(v)$, or $\mathit{color}(v) = \mathit{color}(t)$, then routing proceeds as above. Otherwise, v performs a handshake with a node $w \in \mathsf{ball}(v)$ with $\mathit{color}(w) = \mathit{color}(t)$ to get the identity of l_t as well as $\mathsf{next}(l_t,t)$. Then v routes the message to l_t, where it is forwarded to a node u_t such that $t \in \mathsf{cluster}(u_t)$. At this point, routing proceeds to t along a shortest path.

Stretch of the new routing scheme $\mathcal{R}$. Let $s,t \in V$ be two arbitrary nodes of the graph. We show that the routing scheme $\mathcal{R}$ routes messages from s to t along a route of length at most $5\,\delta(s,t)$ in general, and along a route of length at most $3\,\delta(s,t)$ whenever using handshaking. As we already observed, if $t \in L \cup \mathsf{cluster}(v) \cup \mathsf{ball}(v)$, then the message is routed to t along a shortest path, i.e., with stretch 1. Otherwise, we consider separately whether the color of s is the same as the color of t, or not.

Assume first that $\mathit{color}(t) = \mathit{color}(s)$. Then the message is routed towards l_t along a shortest path, then from l_t to t along a shortest path. The length ℓ of this route satisfies $\ell = \delta(s,l_t) + \delta(l_t,t)$. By the triangle inequality, we get that $\ell \leq \delta(s,t) + 2\delta(l_t,t)$. Since $t \notin \mathsf{cluster}(s)$, we get $\delta(s,t) \geq \delta(t,l_t)$. Therefore $\ell \leq 3\,\delta(s,t)$. We are left with the case where $\mathit{color}(s) \neq \mathit{color}(t)$. Observe first that, with handshaking, the route from s to t will be exactly as the one described in the case $\mathit{color}(s) = \mathit{color}(t)$, resulting in a stretch 3. This completes the proof that $\mathcal{R}$ achieves a stretch 3 with handshaking. Without handshaking, the message is forwarded along a shortest path to an arbitrary node $w \in \mathsf{ball}(v)$ having the same color as t, then from w to l_t along a shortest path, and finally from l_t to t along a shortest path. This route is of length $\delta(s,w) + \delta(w,l_t) + \delta(l_t,t) \leq \delta(s,w) + \delta(w,s) + \delta(s,l_t) + \delta(l_t,t) \leq 2\,\delta(s,w) + 3\,\delta(s,t) \leq 5\,\delta(s,t)$, as desired.

Local Verification of $\mathcal{R}$. We show how to verify $\mathcal{R}$ with a verification algorithm VERIF using certificates on $\widetilde{O}(\sqrt{n})$ bits. Let us define the certificates given to nodes when the routing tables are correctly set as specified by $\mathcal{R}$. For each color c, $1 \leq c \leq \sqrt{n}$, let B_c be the number of bits for encoding the set Dir_c, and let $r = \max_{1 \leq c \leq \sqrt{n}} B_c$. We have $r = \widetilde{O}(\sqrt{n})$. Let $f_1, \ldots, f_k$ be $k = \Theta(\log n)$ hash functions, where each one is mapping sequences of at most r bits onto a single bit. More specifically, each function f_i, $1 \leq i \leq k$, is described as a sequence $f_{i,1}, \ldots, f_{i,r}$ of r bits. Given a sequence $D = (d_1, \ldots, d_\ell)$ of $\ell \leq r$ bits, we set $f_i(D) = (\sum_{j=1}^{\ell} f_{i,j}\, d_j) \bmod 2$. Hence, if the r bits describing f_i are chosen independently

uniformly at random, then, for every two ℓ-bit sets D and D', we have [25]: $D \neq D' \Rightarrow \Pr[f_i(D) = f_i(D')] = 1/2$. Therefore, $D \neq D' \Rightarrow \Pr[(f_i(D) = f_i(D),\ i = 1, \ldots, k)] = 1/2^k$. That is, if $k = \beta\lceil \log_2 n \rceil$ with $\beta > 1$, applying the functions $f_1, \ldots, f_k$ to both sets D and D' enables to detect that they are distinct, with probability $1 - 1/n^\beta$.

Each node $v \in V$ stores the certificate composed of the following fields:

- the distances $\{\delta(v, l) : l \in L\}$;
- the distances $\{\delta(v, u) : u \in \mathsf{ball}(v)\}$;
- the set $\big\{\big(\delta(v, u), l_u, \delta(u, l_u)\big) : u \in \mathsf{cluster}(v)\big\}$;
- the set of k hash functions $f_1, \ldots, f_k$;
- the set $\big\{\big(i, c, f_i(\mathsf{Dir}_c)\big) : 1 \leq i \leq k,\ 1 \leq c \leq \sqrt{n}\big\}$.

The first three entries, as well as the last entry, are clearly on $\widetilde{O}(\sqrt{n})$ bits, because of the sizes of L, $\mathsf{cluster}(v)$, and $\mathsf{ball}(v)$. Regarding the fourth entry, each function f_i is described by a sequence of $\widetilde{O}(\sqrt{n})$ random bits.

The verification algorithm VERIF then proceeds as follows. In a way similar to the proof of Theorem 2, we denote by $H(v)$, $L(v)$, $C(v)$, $B(v)$, $D(v)$, and $\{N(v, t) : t \in L(v) \cup C(v) \cup B(v)\}$ the content of the routing table of v. These entries are supposed to be the hash function *color*, the set of landmarks, the cluster of v, the ball of v, the set $\mathsf{Dir}_{\mathrm{color}(v)}$, and the set of next-pointers given to v, respectively. We also denote by d the distance given in the certificates. That is, node v is given a set $\{d(v, t) : t \in L(v) \cup B(v)\}$, and a set $\{(d(v, t), l_t, d(t, l_t)) : t \in C(v)\}$ where l_t is supposed to be the node in $L(v)$ closest to $t \in C(v)$.

We also denote by $F_1^v, \ldots, F_k^v$ the hash functions given to v in its certificate, and by $F(v) = \{F_{i,c}^v : 1 \leq i \leq k,\ 1 \leq c \leq \sqrt{n}\}$ the set of $O(\sqrt{n} \log n)$ hash values in the certificates. Of course, if a node does not have a table and a certificate of these forms, then it outputs *reject*. So, we assume that all tables and certificates have the above formats.

Clusters, balls and landmarks (including distances, ports, sizes, etc.) are checked exactly as in Section 3 for the routing scheme in [24]. So, in particular, ignoring the colors and ignoring the minimality of the landmarks' identities, we can assume that, for every node v, $L(v) = L$, $C(v) = \mathsf{cluster}(v)$ and $B(v) = \mathsf{ball}(v)$, and, for every node $u \in L \cup \mathsf{cluster}(v) \cup \mathsf{ball}(v)$, we have $d(v, u) = \delta(v, u)$, and $N(v, u) = \mathsf{next}(v, u)$ ignoring the minimality of that port number. To check the remaining entries in the routing tables (as well as the previously ignored colors and minimality criteria for the landmarks and the next-pointers), VERIF performs the following sequence of steps. At every step, if the test is not passed at some node, then VERIF outputs *reject* at this node, and stops. Otherwise, it goes to the next step. If all tests are passed at a node, that node outputs *accept*. Node v checks that:

1. l_v has the smallest identity among all landmarks closest to v;
2. it has the same hash function $H(v)$ as its neighbors, that it has one node of each color in $B(v)$, that v appears in $D(v)$, and that all nodes appearing in $D(v)$ have the same color as v;
3. if $l_v \neq v$ then there exists at least one neighbor u on a shortest path between v and l_v with $N(l_v, v) = N(l_v, u)$; for every neighbor u on the shortest path between v and l_v (implying $l_u = l_v$) with $u \neq l_v$, $N(l_v, v) \leq N(l_u, u)$; and if l_v is a neighbor of v on a shortest path between l_v and v, $N(l_v, v) = \mathsf{port}_{l_v}(\{l_v, v\})$;
4. for every $u \in L \cup B(v) \cup C(v)$, the port number $N(v, u)$ is the smallest among all the ports of edges incident to v on a shortest path between v and u;
5. it has the same hash functions $F_1^v, \ldots, F_k^v$, as its neighbors;
6. $F_i^v(D(v)) = F_{i,H(v)}^v$ for every $i = 1, \ldots, k$;
7. the hash value $F_{i,c}^v$, $1 \leq i \leq k$, $1 \leq c \leq \sqrt{n}$, are identical to the ones of their neighbors.

If v passes all the above tests, then v outputs *accept*, else it outputs *reject*.

We now establish the correctness of this local verification algorithm. First, by construction, if all tables are set according to the specification of $\mathcal{R}$, then every node running the verification algorithm with the appropriate certificate will face no inconsistencies with its neighbors, i.e., all the above tests are passed, leading every node to accept, as desired. So, it remains to show that if some tables are not what they should be according to $\mathcal{R}$, then, no matter the certificates assigned to the nodes, at least one node will fail one of the tests with high probability.

If all nodes pass the test of Step 1, then we are guaranteed that not only $L(v) = L$, but also that l_v is indeed the appropriate landmark of v. If all nodes pass the test of Step 2, then it must be the case that $H(v) = \mathit{color}(v)$, that $B(v) = \mathsf{ball}(v)$ with the desired coloring property, and that $D(v) \subseteq \mathsf{Dir}_{\mathrm{color}(v)}$. By Lemma 6, we get that if all nodes pass Steps 3 and 4, then $N(v,u) = \mathsf{next}(v,u)$ where the minimality condition is satisfied. Let us assume that there exists a pair of nodes (u,v) with same color such that $(u, l_u, \mathsf{next}(l_u,u)) \in \mathsf{Dir}_{\mathrm{color}(v)} \setminus D(v)$. By the previous steps, we know that $(u, l_u, \mathsf{next}(l_u,u)) \in D(u)$. Hence, $D(u) \neq D(v)$. On the other hand, if Step 5 is passed by all nodes, then all nodes agree on a set of $k = \Theta(\log n)$ hash function $f_1, \ldots, f_k$. Therefore, assuming that these functions are set at random, we get that, with high probability, there exists at least one function f_i, $1 \leq i \leq k$, such that $f_i(D(v)) \neq f_i(D(u))$. If all nodes pass Step 6, then in particular $F_i^v(D(v)) = F_{i,c}^v$, and $F_i^u(D(u)) = F_{i,c}^u$ where $c = \mathit{color}(v) = \mathit{color}(u)$. We know that $F_{i,c}^v = f_i(D(v))$ and $F_{i,c}^u = f_i(D(u))$, which implies that $F_{i,c}^v \neq F_{i,c}^u$ with high probability, which will be detected at Step 7 by two neighboring nodes in the network. This completes the proof of Theorem 4. ◀

▶ Remark. We have seen that the stretch-3 name-independent routing scheme from [2] does not seem to be locally checkable with certificates of $\widetilde{O}(\sqrt{n})$ bits. Our new name-independent routing scheme $\mathcal{R}$ is locally checkable with certificates of $\widetilde{O}(\sqrt{n})$ bits, but it has stretch 5. We can show that, as for the scheme in [2], the routing scheme in [3] do not appear to be locally checkable with certificates of $\widetilde{O}(\sqrt{n})$ bits. Moreover, handshaking, which may allow that scheme to become locally verifiable with small certificates, does not reduce its stretch (for details, please refer to the full version of this paper [6]).

6 Conclusion and Further Work

We have shown that it is possible to verify routing schemes based on tables of size $\widetilde{O}(\sqrt{n})$ bits using certificates with sizes of the same order of magnitude as the space consumed by the routing tables. The stretch factor is preserved, but to the cost of using handshaking mechanisms for name-independent routing. We do not know whether there exists a stretch-3 name-independent routing scheme, with tables of size $\widetilde{O}(\sqrt{n})$ bits, that can be verified using certificates on $\widetilde{O}(\sqrt{n})$ bits. Our new routing scheme, which is verifiable with certificates on $\widetilde{O}(\sqrt{n})$, has stretch 3 only if using handshaking (otherwise, it has stretch 5). Moreover, the certification of our routing scheme is probabilistic, and it would be of interest to figure out whether deterministic certification exists for some stretch-3 name-independent routing scheme, with tables and certificates on $\widetilde{O}(\sqrt{n})$ bits. It could also be of interest to figure out whether there exists a verification scheme using certificates of size $\widetilde{O}(n^c)$ bits, with $c < \frac{1}{2}$.

Interestingly, our result for stretch-3 name-dependent routing can be extended to larger stretches. Namely, by using the same techniques as for stretch 3, we can show that the family of routing schemes in [24] using, for every $k \geq 1$, tables of size $\widetilde{O}(n^{1/k})$ with stretch at most $4k-5$ (or $2k-1$ using handshaking) are verifiable with certificates of size $\widetilde{O}(n^{1/k})$ (see [6] for more details). However, we do not know whether such a tradeoff between table sizes and

stretches can be established for verifiable *name-independent* routing schemes. Specifically, are the existing families of name-independent routing schemes using, for every $k \geq 1$, tables of size $\widetilde{O}(n^{1/k})$ with stretch at most $O(k)$, verifiable with certificates of size $\widetilde{O}(n^{1/k})$? If not, is it possible to design a new family of verifiable name-independent routing schemes satisfying the same size-stretch tradeoff?

Acknowledgements. The authors are thankful to Cyril Gavoille and Laurent Viennot for discussions about the content of this paper.

References

1. Ittai Abraham, Cyril Gavoille, and Dahlia Malkhi. On space-stretch trade-offs: upper bounds. In *SPAA 2006: Proceedings of the 18th Annual ACM Symposium on Parallelism in Algorithms and Architectures, Cambridge, Massachusetts, USA, July 30 - August 2, 2006*, pages 217–224, 2006.
2. Ittai Abraham, Cyril Gavoille, Dahlia Malkhi, Noam Nisan, and Mikkel Thorup. Compact name-independent routing with minimum stretch. In *SPAA 2004: Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, June 27-30, 2004, Barcelona, Spain*, pages 20–24, 2004.
3. Marta Arias, Lenore Cowen, Kofi A. Laing, Rajmohan Rajaraman, and Orjeta Taka. Compact routing with name independence. In *SPAA 2003: Proceedings of the Fifteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, June 7-9, 2003, San Diego, California, USA (part of FCRC 2003)*, pages 184–192, 2003.
4. Baruch Awerbuch, Amotz Bar-Noy, Nathan Linial, and David Peleg. Compact distributed data structures for adaptive routing (extended abstract). In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 479–489, 1989.
5. Baruch Awerbuch and David Peleg. Sparse partitions (extended abstract). In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume II*, pages 503–513, 1990.
6. Alkida Balliu and Pierre Fraigniaud. Certification of compact low-stretch routing schemes. In *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, 2017.
7. Mor Baruch, Pierre Fraigniaud, and Boaz Patt-Shamir. Randomized proof-labeling schemes. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 315–324, 2015.
8. Shiri Chechik. Compact routing schemes with improved stretch. In *ACM Symposium on Principles of Distributed Computing, PODC'13, Montreal, QC, Canada, July 22-24, 2013*, pages 33–41, 2013.
9. Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno. *Cryptography Engineering: Design Principles and Practical Applications*. Wiley Publishing, 2010.
10. Laurent Feuilloley and Pierre Fraigniaud. Survey of distributed decision. *Bulletin of the EATCS*, 119, 2016.
11. Pierre Fraigniaud and Cyril Gavoille. Routing in trees. In *Automata, Languages and Programming, 28th International Colloquium, ICALP 2001, Crete, Greece, July 8-12, 2001, Proceedings*, pages 757–772, 2001.
12. Pierre Fraigniaud and Cyril Gavoille. A space lower bound for routing in trees. In *STACS 2002, 19th Annual Symposium on Theoretical Aspects of Computer Science, Antibes - Juan les Pins, France, March 14-16, 2002, Proceedings*, pages 65–75, 2002.

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

14 Klaus-Tycho Förster, Thomas Lüdi, Jochen Seidel, and Roger Wattenhofer. Local Checkability, No Strings Attached: (A)cyclicity, Reachability, Loop Free Updates in SDNs. In *Theoretical Computer Science (TCS)*, November 2016.

15 Cyril Gavoille. Routing in distributed networks: overview and open problems. *SIGACT News*, 32(1):36–52, 2001.

16 Cyril Gavoille and Marc Gengler. Space-efficiency for routing schemes of stretch factor three. *J. Parallel Distrib. Comput.*, 61(5):679–687, 2001.

17 Cyril Gavoille and David Peleg. Compact and localized distributed data structures. *Distributed Computing*, 16(2-3):111–120, 2003.

18 Cyril Gavoille and Stephane Perennes. Memory requirements for routing in distributed networks (extended abstract). In *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing, Philadelphia, Pennsylvania, USA, May 23-26, 1996*, pages 125–133, 1996.

19 Mika Göös and Jukka Suomela. Locally checkable proofs in distributed computing. *Theory of Computing*, 12(1):1–33, 2016.

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

21 David Peleg and Eli Upfal. A tradeoff between space and efficiency for routing tables (extended abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 43–52, 1988.

22 Nicola Santoro and Ramez Khatib. Labelling and implicit routing in networks. *Comput. J.*, 28(1):5–8, 1985.

23 Stefan Schmid and Jukka Suomela. Exploiting locality in distributed SDN control. In *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, 2013.

24 Mikkel Thorup and Uri Zwick. Compact routing schemes. In *SPAA*, pages 1–10, 2001.

25 Andrew Chi-Chih Yao. Some complexity questions related to distributive computing (preliminary report). In *Proceedings of the 11h Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1979, Atlanta, Georgia, USA*, pages 209–213, 1979.