Shortest Paths in a Hybrid Network Model*

John Augustine† Kristian Hinnenthal‡ Fabian Kuhn§ Christian Scheideler¶

Philipp Schneider‖

Abstract

We introduce a communication model for *hybrid networks*, where nodes have access to two different communication modes: a *local* mode where (like in traditional networks) communication is only possible between specific pairs of nodes, and a *global* mode where (like in overlay networks) communication between any pair of nodes is possible. Typically, communication over short-range connections is cheaper and can be done at a much higher rate than communication via the overlay network. Therefore, we are focusing on the LOCAL model for the local connections where nodes can exchange an unbounded amount of information per round. For the global communication we assume the so-called *node-capacitated clique model*, where in each round every node can exchange $O(\log n)$-bit messages with $O(\log n)$ arbitrary nodes.

We explore the impact of hybrid communication on the complexity of distributed algorithms by studying the problem of computing *shortest paths* in the graph given by the local connections. We present the following results. For the all-pairs shortest paths problem, we show that an exact solution can be computed in time $\tilde{O}\big(n^{2/3}\big)$ and that approximate solutions can be computed in time $\tilde{\Theta}\big(\sqrt{n}\big)$ but not faster. For the single-source shortest paths problem an exact solution can be computed in time $\tilde{O}\big(\sqrt{\mathsf{SPD}}\big)$, where SPD denotes the shortest path diameter. Furthermore, a $\big(1+o(1)\big)$-approximate solution can be computed in time $\tilde{O}(n^{1/3})$. Finally, we show that for every constant $\varepsilon > 0$, it is possible to compute an $O(1)$-approximate solution in time $\tilde{O}(n^{\varepsilon})$.

1 Introduction

Many existing communication networks exploit a combination of multiple communication modes to maximize cost-efficiency and throughput. As a prominent example, hybrid datacenter networks combine high-speed optical or wireless circuit switching technologies with traditional electronic packet switches to offer higher throughput at lower cost [14, 20]. In the Internet, dynamic multipoint VPNs can be set up to connect different branches of an organization by combining leased lines (offering them quality-of-service guarantees for their mission-critical traffic) with standard, best-effort VPN connections (for their lower-priority traffic) [34]. Alternatively, an organization may also set up a so-called hybrid WAN by combining their own communication infrastructure with connections via the Internet [36]. Finally, the emerging 5G standard promises to allow handheld devices to not only communicate via the cellular infrastructure, but also directly with other devices via their wireless interface. This allows them to set up a hybrid network consisting of connections via base stations as well as device-to-device (D2D) connections [26], which is particularly interesting for vehicular networks.

Despite the advantages that have been experienced with hybrid communication networks in practice, rigorous theoretical research on hybrid networks is still in its infancy. In this paper, we propose a simple model for hybrid networks. In our model, we assume that each node has two different communication modes: a *local* communication mode that allows it to send messages along each of its edges in the given (private, leased, trusted, or ad-hoc) communication network, and a *global* communication mode, which allows a node to send messages to any node in the network, but to only exchange a limited number of messages in each round using this mode.

We explore the power of the model for a fundamental problem in graph theory as well as communication networks: computing shortest paths. While shortest paths problems are interesting from a purely theoretical standpoint, this class of problems has concrete applications in practice (e.g., IP-routing). In particular, we consider the problem of computing all-pair and single-source shortest paths (exactly and approximately) in the local network (weighted and unweighted). We demonstrate that by making use of both local and global communication, we can achieve significant runtime improvements for this class of problems compared to using local or global edges alone, which highlights the importance of exploiting hybrid communication capabilities of modern networks.

Before we present our detailed results in Section 1.2, we formally introduce our model. We conclude this section with a discussion of related work in Section 1.3.

*This work is partially supported by the German Research Foundation (DFG) within the CRC 901 "On-The-Fly Computing" (project number 160364472-SFB901).

†IIT Madras, India. augustine@iitm.ac.in

‡Paderborn University, Germany. krijan@mail.upb.de

§University of Freiburg, Germany. kuhn@cs.uni-freiburg.de

¶Paderborn University, Germany. scheideler@upb.de

‖Uni. of Freiburg, Germany. philipp.schneider@cs.uni-freiburg.de

1.1 Hybrid Communication Model

We assume that we are given a fixed set V of n nodes[1] that are connected via two kinds of edges: *local* edges and *global* edges. The local edges form a fixed, undirected, weighted graph $G = (V, E, w)$, where the edge weights are given by $w : E \to \{1, \ldots, W\} \subseteq \mathbb{N}$ for some W that is at most polynomial in n. Thus, every weight and length of any shortest path can be represented using $O(\log n)$ bits. The graph G is said to be *unweighted* if $w : E \to \{1\}$. The global edges form a clique, i.e., every node can potentially send a message to any other node with the help of a global edge. We assume that each node u has a unique identifier $\text{id}(u)$. For simplicity, we assume that the node identifiers are $1, \ldots, n$.[2]

We use the standard synchronous message passing model, where time is divided into synchronous *rounds*, and in each round every node can send messages of size $O(\log n)$ to other nodes using its local and global edges. In the most general form of our model, the number of messages that can be sent with either communication mode is restricted by parameters λ and γ: the *local capacity* λ is the maximum number of messages that can be sent over each local edge in a round, and the *global capacity* γ is the maximum number of messages any node can send and receive via global edges in a round. When in some round more than λ (or γ) messages are sent over an edge (or to a node, respectively), we assume that an adversary delivers an arbitrary subset of these messages and drops the other messages. All of our algorithms ensure that with high probability[3], a node never sends or receives too many messages.

Note that whereas λ imposes a bound on the number of messages that can be sent over each *edge*, γ effectively restricts the amount of global communication at each *node*. This modeling choice is motivated by the idea that local communication rather relates to *physical* networks, where an edge corresponds to a physical connection (e.g., cable- or ad-hoc networks), whereas global communication primarily captures aspects of *logical* networks that are formed as an overlay on top of some shared physical infrastructure. For appropriate choices of λ and γ, our model captures various established network models: LOCAL ($\lambda = \infty, \gamma = 0$), CONGEST ($\lambda = O(1), \gamma = 0$), *congested clique*[4] ($\lambda = O(1), \gamma = 0$ where G is a clique but the input graph is given separately), and the recently introduced *node-capacitated clique* model ($\lambda = 0, \gamma = O(\log n)$) [4].

In order to demonstrate the power of combining local and global communication, we will focus on the variant of the model in which local edges are fully uncapacitated ($\lambda = \infty$), and global communication is heavily restricted ($\gamma = O(\log n)$). Thus, our model is a combination of the most permissive LOCAL model for local edges and the very restrictive node-capacitated clique model for the global edges, which makes it particularly clean and well-suited to investigate the power of hybrid networks from a theoretical perspective. Moreover, we believe that the practical relevance of this model is justified by the fact that direct connections between devices are typically highly efficient and offer a large bandwidth at comparatively low cost, whereas communication over a shared global communication network such as the Internet, satellites, or the cellular network, is costly and typically offers only a comparatively small data rate.

We remark that any choice of γ in the range from $\Theta(1)$ to $\Theta(\log^c n)$ would not significantly change our upper bounds, as it would only affect them by polylogarithmic factors, which we mostly neglect by using the $\widetilde{O}(\cdot)$-notation.[5] The maximally permissive choice of $\lambda = \infty$ is mainly for proving lower bounds (which we do for APSP) and most of our algorithms make no overly excessive use of it. In fact, we show that most of our algorithms work for some λ between $\Theta(1)$ and $\Theta(n)$ (with one exception, c.f., Table 1).

1.2 Contributions

The overarching goal of this paper is to achieve significantly faster solutions for shortest path problems than what would be possible if only either the local or the global network could be used. Note that by just using the local network (LOCAL model), all graph problems can trivially be solved in time at most D, where D is the diameter of the graph G. Therefore, any upper bound $\widetilde{O}(f)$ we give in the following for some function f, can be read as $\widetilde{O}\big(\min(D, f)\big)$. Shortest path problems also clearly have an $\Omega(D)$ lower bound using only the local network (note that $D = \Omega(n)$ in the worst case). Our objective is to understand to what extent a limited amount of global communication (given by the global network) helps in solving shortest path problems faster. To that end, we will briefly discuss our contributions. A summary of our results is given in Table 1.

Token Dissemination. First we consider the *token dissemination* problem. It represents the task of

[1]Throughout the paper, we assume that $n \geq n_0$ for a sufficiently large constant n_0.

[2]Although assuming that ids are from 1 to n is a fairly strong assumption (which allows each node to sample nodes from V at random), all our results can be obtained in the same way if nodes have arbitrary $O(\log n)$-bit IDs and have access to a sampling service that allows them to contact a node that is chosen approximately uniformly at random via a global edge.

[3]An event holds with high probability (w.h.p.) if it holds with probability at least $1 - \frac{1}{n^c}$ for an arbitrary but fixed constant $c > 0$.

[4]The congested clique model refers to the unicast variant, i.e., the CONGEST model on a clique topology.

[5]The $\widetilde{O}(\cdot)$-notation suppresses factors poly-logarithmic *in* n.

APSP				**SSSP**			
Approx.	Weights	Complexity	Local Capacity[†]	Approx.	Weights	Complexity	Local Capacity[†]
Exact	weighted	$\widetilde{O}(n^{2/3})$	$O(n)$	Exact	weighted	$\widetilde{O}(\sqrt{\mathsf{SPD}})$	$\widetilde{O}(n^2)$
$(1+\varepsilon)$	unweighted	$\widetilde{O}(\sqrt{n/\varepsilon})$	$O(n)$	$(1+\varepsilon)$	weighted	$\widetilde{O}(n^{1/3}/\varepsilon^6)$	$\widetilde{O}(n^{2/3}\varepsilon^6)$
3	weighted	$\widetilde{O}(\sqrt{n})$	$O(n)$	$(1/\varepsilon)^{O(1/\varepsilon)}$	weighted	$\widetilde{O}(n^{\varepsilon})$	$O(1)$
$\Omega(\sqrt{n})$	unweighted	$\widetilde{\Omega}(\sqrt{n})$	∞	$s(n)^‡$	weighted	$s(n)^‡$	$O(1)$

† Local capacity λ for which the corresponding runtime can still be achieved.
‡ $s(n) = 2^{O(\sqrt{\log n \log\log n})} = o(n^c)$ for arbitrary $c > 0$.

Table 1: Overview of the contributions of this paper.

broadcasting a set of tokens of size $O(\log n)$ bits, each of which is initially only known by one node. We develop a protocol, tailored to the hybrid model, which we use as a subroutine throughout the paper. The main idea behind the algorithm is to randomly disseminate the tokens via global edges. This is sufficient for each node to afterwards collect all the tokens in a relatively small neighborhood using local edges. The runtime compares favourably to the respective lower bounds of the problem in case only the global or local network would be available. Specifically, we show an upper bound of $\widetilde{O}(\sqrt{k} + \ell)$, where k is the number of distinct tokens and ℓ is the initial maximum number of tokens per node (see Section 2.1). Note that the lower bound is $\Omega((k+\ell)/\log n)$ if only the global network can be used (since there are nodes that need to send ℓ or receive $\Omega(k)$ tokens, respectively) and $\Omega(n)$ if only the local network is available (for graphs with $D = \Omega(n)$).

All-Pairs Shortest Paths (APSP). Our primitives to solve APSP are based on combining the token dissemination scheme with the classic approach [37] of building skeleton graphs.[6] The basic idea is to sample a set of nodes with some probability $\frac{1}{x}$ and then compute virtual edges among pairs of sampled nodes connected by a path of at most $h \in \widetilde{O}(x)$ hops. We then employ our token dissemination protocol to broadcast (the length of) all virtual edges of the skeleton graph. Token dissemination is also used to make distance information between any node and its skeleton nodes within h hops global knowledge. With the global knowledge gained in this way, any node can compute its distance to any other node with sufficient hop distance.

While the approach of using skeleton graphs is not new, we demonstrate how the amount of work on the local and global network can be balanced (with parameter x), leading to interesting exact and approximate results for APSP. Specifically, in Theorem 2.2, we show that APSP can be solved exactly with running time $\widetilde{O}(n^{2/3})$. Furthermore, we obtain 3-approximate distances in time $\widetilde{O}(\sqrt{n})$ for general graphs (Theorem 2.3) and $(1+\varepsilon)$-approximate distances in time $\widetilde{O}(\sqrt{n/\varepsilon})$ for unweighted graphs (Theorem 2.4). Note that this is significantly better than the $\widetilde{\Omega}(n)$ bound if only either the local or the global network could be used. These bounds immediately follow from the facts that the diameter of the local network might be $\Omega(n)$ and that every node can only receive $O(\log n)$ messages over global edges.

Finally, we complement our upper bounds for APSP in the hybrid model with a lower bound by proving that even for computing an α-approximate solution for some $\alpha = \widetilde{O}(\sqrt{n})$, at least $\widetilde{\Omega}(\sqrt{n})$ rounds are required (Theorem 2.5), showing that our approximate APSP algorithms are tight up to logarithmic factors.

Single-Source Shortest Paths (SSSP). For the SSSP problem, we present three algorithms that require different techniques. In Theorem 2.6, we prove that the SSSP problem can be solved exactly in time $\widetilde{O}(\sqrt{\mathsf{SPD}})$, where the *shortest-path diameter* SPD is the smallest value h such that there exists a shortest path with h many hops between ever pair of nodes in G. Note that $\mathsf{SPD} < n$, since edge weights are non-negative. The algorithm combines the local and global network in the following way. Using the local network, every node learns the graph up to a distance of $2\sqrt{\mathsf{SPD}}$ hops, thus learning the $\sqrt{\mathsf{SPD}}$ neighborhood of any node within $\sqrt{\mathsf{SPD}}$ hops. This knowledge is used to distribute distance information from the source s in an iterative fashion over the global network, where in iteration i, all nodes that have a shortest path to s with $O(i^2)$ hops learn their distance to s. An iteration takes only $\widetilde{O}(1)$ rounds, leveraging a divide-and-conquer approach to distribute the information in a node's neighborhood, and the aggregation protocol of [4]. Note that using only either local or global edges, the best known algorithms

[6]We remark that although some of our algorithms resemble the *hopset* approach (see, e.g., [13]), we never explicitly construct hopsets in the classical sense. Notably, our constructions are not specifically tailored to achieve an overall small hopbound.

require $\Omega(\mathsf{SPD})$ rounds (which is tight for local edges as $D = \mathsf{SPD}$ on unweighted graphs).

We then shift our attention to approximate solutions of the SSSP problem. We give a simple algorithm that simulates the *broadcast congested clique* model (BCC)[7] on a set of sampled skeleton nodes (including the source) using token dissemination. We employ the SSSP algorithm by Becker et al. [7] for said model as a black box to solve SSSP on the skeleton, which allows us to compute a $\big(1+o(1)\big)$ approximation of SSSP in time $\widetilde{O}(n^{1/3})$ (Theorem 2.7).

The third, technically most challenging SSSP algorithm is based on recursively building a hierarchy of $O(\log_\alpha n)$ (for some $\alpha > 1$) *skeleton spanners* (i.e., spanners of skeleton graphs). Roughly speaking, given some skeleton spanner H, we obtain the next coarser skeleton spanner H' by sampling each node of H with probability $1/\alpha$ and computing a spanner with a good stretch on the sampled nodes. As a technical result, we show that given a low arboricity[8] graph H, we can efficiently compute a low arboricity spanner H' of H using only global edges. We show in Theorem 2.8 that by choosing $\alpha = n^\varepsilon$ for some $\varepsilon > 0$, we can compute $(1/\varepsilon)^{O(1/\varepsilon)}$-approximate paths to the source node in time $\widetilde{O}(n^\varepsilon)$. For any constant $\varepsilon > 0$, we get a constant SSSP approximation (albeit with a potentially large constant). Choosing ε to balance time and approximation factor, the algorithm computes a (subpolynomial) $2^{O(\sqrt{\log n \log\log n})}$-approximate SSSP solution in the same time.

1.3 Related Work

In the theory area, only very few results are known so far on hybrid networks. In order to provide the reader with a good perspective over the field, we widen our focus to models that overlap with our generic hybrid model.

Hybrid Communication Networks. In the systems area, research on hybrid networks has mostly focused on wireless mesh networks (see, e.g., [3] for a recent survey), with a plethora of competing schemes for routing packets in such a network, though many of them do not exploit the hybrid communication capabilities of such networks. Hybrid communication has also been studied in the context of data centers demonstrating that it can significantly improve their cost-effectiveness and performance (e.g., [14, 20]).

On the theoretical side, Jung et al. [24] studied the problem of finding near-shortest routing paths in adhoc networks satisfying certain properties by combining communication via ad-hoc connections with communication via the cellular infrastructure. Furthermore, Foerster et al. [15] investigated the computational complexity of exploiting a hybrid infrastructure in data centers.

Global Communication Networks. More recently, distributed models that are closely related to our global communication mode have been considered. For example, the work of Gmyr et al. [19] implies that making use of global edges with the same constraints as in our model significantly improves the ability to monitor properties of the network formed by the local edges. Furthermore, [4] introduces the *node-capacitated clique* model, which is identical to our global communication mode (but without local edges). They present distributed algorithms for various fundamental graph problems, including computation of MSTs, BFS trees, maximal independent sets, maximal matchings, or vertex colorings of the given input graph. Their BFS tree construction can be used to solve the SSSP problem for unweighted graphs of bounded arboricity in $\widetilde{O}(D)$ time.

For the *congested clique model*,[9] research has been very active over the last few years (see, e.g., [12, 25, 27, 31] for a small subset of the work). In the context of shortest path problems, Lenzen et al. [7] presented a $(1+\varepsilon)$-approximation algorithm for the SSSP problem that runs in time polylog n and Nanongkai [32] gave a $\big(2+o(1)\big)$-approximation algorithm for the APSP problem with runtime $\widetilde{O}(\sqrt{n})$. The algorithms of [7] and [32] even work for the broadcast variant of the congested clique where in each round, every node has to send the same $O(\log n)$-bit message to all other nodes.[10] Shortest path problems can also be approached by performing matrix multiplications efficiently [11]. Then, APSP can be solved exactly in time $\widetilde{O}(n^{1/3})$, and a $\big(1+o(1)\big)$-approximation can be found in time $O(n^{0.158})$. A recent result shows that a $\big(2+o(1)\big)$-approximation can even be computed in time polylog n [10].

Local Communication Networks. Shortest path problems have been intensely studied in standard distributed communication models, most importantly in the $\mathsf{CONGEST}$ model. Some of our algorithms for the hybrid network model employ ideas that have been developed in this context. For the SSSP problem, Das Sarma et al. [35] showed that any distributed

[7]In the BCC, in each round every node can broadcast one $O(\log n)$-bit message per round to all other nodes.

[8]The arboricity of a graph is the minimum number of forests required to cover all edges.

[9]Note that the hybrid model contains the congested clique model as special case (see Section 1.1). Our global network can simulate the congested clique for $\gamma = O(n)$ (though this is an even more powerful model). However, for the specific case $\gamma = O(\log n)$ considered in this paper the results for the congested clique are not helpful, since it is too costly to emulate these algorithms.

[10]We will exploit this fact for one of our results (Section 5.2), by showing that we can simulate the broadcast congested clique model on a small subset of nodes in the hybrid model.

approximation algorithm has a runtime of $\widetilde{\Omega}(\sqrt{n}+D)$ for any constant approximation ratio. Following the publication of this lower bound, there has been a series of papers that attempt to obtain algorithms that get close to the lower bound (see, e.g., [28, 32, 21]), culminating in the work of Becker et al. [7], which gives an algorithm that computes a $(1+\varepsilon)$-approximate SSSP solution in time $\widetilde{O}(\sqrt{n}+D)$. For the exact SSSP problem, no better upper bound than $O(n)$ was known until two years ago when Elkin [13] presented an algorithm with a runtime of $\widetilde{O}(n^{2/3}D^{1/3}+n^{5/6})$. This was further improved by Ghaffari and Li [18] and by Forster and Nanongkai [16], who presented two protocols for polynomially bounded edge weights, one with runtime $\widetilde{O}(\sqrt{n \cdot D})$ and one with runtime $\widetilde{O}(\sqrt{n}D^{1/4}+n^{3/5}+D)$.

For the APSP problem, a deterministic $\big(1+o(1)\big)$-approximation algorithm with runtime $\widetilde{O}(n)$ is known, as well as a nearly matching lower bound of $\widetilde{\Omega}(n)$ that holds for randomized poly(n)-approximation algorithms, even when $D = O(1)$ [28, 29, 32]. The complexity of the exact unweighted version was shown to be $\widetilde{\Theta}(n)$ [30, 22, 17, 33, 1]. For the exact weighted version, the first improvement over the naive $O(m)$-time algorithm was due to Huang et al. [23], who presented a randomized $\widetilde{O}(n^{5/4})$-time algorithm that bears similarity to our approaches based on skeleton nodes (which they call "centers"). Subsequently, Bernstein and Nanongkai [8] came up with a randomized $\widetilde{O}(n)$-time algorithm, therefore also this case is now settled up to polylog(n) factors. The best deterministic algorithm for the weighted APSP problem is due to Agarwal et al. [2] and has a runtime of $\widetilde{O}(n^{3/2})$ using a technique based on "blocker sets" quite similar to ours based on skeleton nodes.

2 Overview

In this section we provide the reader with an intuitive explanation of our core concepts without the in-depth technical details of the subsequent sections. Additionally, we will state our main theorems and sketch some proof ideas. Let us start by introducing some basic definitions.

2.1 Preliminaries and Problem Definitions.

The *distance* between any two nodes $u, v \in V$ of a graph $G = (V, E)$ is defined as

$$d_G(u,v) := \min_{u\text{-}v\text{-path } P} w(P),$$

where $w(P) = \sum_{e \in P} w(e)$ denotes the length of a path $P \subseteq E$. A path between two nodes with smallest length is called a *shortest path*. The *hop-distance* between two nodes u and v is defined as

$$hop_G(u,v) := \min_{u\text{-}v\text{-path } P} |P|,$$

where $|P|$ denotes the number of edges (or *hops*) of a path P. Similarly, let the *shortest path hop-distance* be

$$sph_G(u,v) := \min_{\text{shortest } u\text{-}v\text{-path } P} |P|.$$

The *diameter* of G is defined as

$$D(G) := \max_{u,v \in V} hop_G(u,v)$$

Let the *h-limited distance* from u to v

$$d_{h,G}(u,v) := \min_{\substack{u\text{-}v\text{-path } P \\ |P| \le h}} w(P).$$

If there is no u-v path P with $|P| \le h$ let $d_{h,G}(u,v) := \infty$. The *shortest-path diameter* $\mathsf{SPD}(G)$ is the minimum number such that $d_{\mathsf{SPD}(G),G}(u,v) = d_G(u,v)$ for all $u, v \in V$. Whenever the graph G is clear from the context, we drop the subscript G in the above notations. In this paper, we consider the following shortest-paths problems in G.

All-Pairs Shortest Paths Problem (APSP). Every node $u \in V$ has to learn $d(u,v)$ for all $v \in V$. In the α-approximate APSP problem for some $\alpha > 1$, every node $u \in V$ has to learn values $\tilde{d}(u,v)$ such that $d(u,v) \le \tilde{d}(u,v) \le \alpha \cdot d(u,v)$ for all $v \in V$.

Single-Source Shortest Paths Problem (SSSP). There is a source $s \in V$ and every node $u \in V$ has to learn $d(u,s)$. In the α-approximate SSSP problem for some $\alpha > 1$, every node $u \in V$ has to learn $\tilde{d}(u,s)$ such that $d(u,s) \le \tilde{d}(u,s) \le \alpha \cdot d(u,s)$.

In order to solve shortest paths problems efficiently, we also show how to solve the *(k,ℓ)-token dissemination problem ((k,ℓ)-TD)*. Here we are given a set of k tokens each of size $O(\log n)$-bits that need to be learned by all nodes $v \in V$. Initially, each token is known by one node and no node initially possesses more than ℓ tokens. As a byproduct of our exact SSSP algorithm, we solve the *h-limited k-source shortest paths problem ((h,k)-SSP)*, in which there is a set $S \subseteq V$ of k sources and a parameter $h \ge 1$ and every node $u \in V$ with $sph(s,v) \le h$ has to learn $d(u,s)$ for every $s \in S$.

2.2 Token Dissemination.

The first tool that we are introducing solves the token dissemination problem (k,ℓ)-TD. The algorithm consists of four steps. First, we balance the number of tokens per node. Each node redistributes its tokens randomly via global edges for $O(\ell)$ rounds such that afterwards, w.h.p., each node has at most $\widetilde{O}(k/n)$ tokens to take care of. This first step eliminates the dependency on ℓ in the subsequent steps.

Second, if $k \ll n$, we copy each token to $\widetilde{O}(n/k)$ nodes, which allows to speed up the subsequent third step. We increase the number of copies of each token in the network in an exponential fashion in $\log(n/k)$ phases. In each phase, every node sends two copies of the tokens it received in the previous phase to random nodes via global edges. Note that this works because the total number of copies remains in $\widetilde{O}(n)$ and therefore the contention on the global network is not too high.

Third, each node sends the tokens it knows so far via global edges to a random subset of V, so that afterwards each node possesses a given token with probability at least $1/x$. This takes only $O(k/x)$ rounds, relying on the fact that for $k \ll n$ we have $\widetilde{O}(n/k)$ nodes per token helping to disseminate it from the previous step. Afterwards, the $\widetilde{\Omega}(x)$-neighborhood (w.r.t. local edges) of any node contains all tokens, w.h.p.

Fourth, the local edges are used to learn the tokens of $\widetilde{\Omega}(x)$ nodes in the neighborhood of any given node, which takes $\widetilde{O}(x)$ rounds. The parameter x signifies the trade-off between the running time of the third and fourth step and is optimized accordingly ($x \in \widetilde{O}(\sqrt{k})$). In Section 3 we give the details of the algorithm (Algorithm 1) and we provide a full proof of Theorem 2.1.

Theorem 2.1. *There is an algorithm that solves (k,ℓ)-TD on connected graphs in $\widetilde{O}(\sqrt{k}+\ell)$ rounds, w.h.p.*

2.3 Upper Bounds for Exact APSP. The first step to solve APSP is to construct an overlay graph $S = (M, E_S)$ on G that we call a *skeleton* [37] and whose nodes $M \subseteq V$ are obtained by marking nodes of V uniformly at random (with probability $1/x$, for some optimization parameter x). Two nodes in M have an edge if their hop-distance is at most h. The weight of such an edge is the h-limited distance between its endpoints. To compute E_S, we explore a (small) h-hop-neighborhood around every node in the local network. Since we choose $h = \widetilde{\Theta}(x)$, this takes $\widetilde{O}(x)$ rounds. Afterwards, each node knows the h-limited distance between it and all other nodes.

After the local exploration, we do not have to worry about pairs of nodes for which a shortest path of at most h hops exists, since for those pairs the h-limited distances equal the true distance. For pairs $u, v \in V$, for which all shortest u-v-paths have *more* than h hops, we show that on one such a path there is a skeleton node within every h hops w.h.p. (Lemma 4.1). This is particularly helpful since these pairs can now compute their distance if they have knowledge of the distance information of the (sparse) set of skeleton nodes.

The expected size of M is $|M| \in \widetilde{O}(n/x)$, and we show that the weights of all edges E_S can be broadcast to the whole network in $\widetilde{O}(n/x)$ with the methods of Section 3 (token dissemination). Equipped with that information about E_S, each node can locally compute the distance matrix D^S of the skeleton S, and we show that distances among nodes in S equal those in G. Finally, we disseminate the h-limited distances between pairs $M \times V \setminus M$, which can be done in $\widetilde{O}(n/\sqrt{x})$. With this information, all nodes know the distance matrix D' containing said h-limited distances between pairs $M \times V \setminus M$. Then any node in V can locally compute the complete distance matrix D^G of G as follows (we set $D'_{uv} = 0$ if $u, v \in M$):

$$D^G_{uv} = \min\Big(d_h(u,v), \min_{u',v' \in M}\big(D'_{uu'} + D^S_{u'v'} + D'_{vv'}\big)\Big).$$

For $x \in [1..n]$ the running time for the exploration via local edges is $\widetilde{O}(x)$ and $\widetilde{O}(n/\sqrt{x})$ for the dissemination of the distance matrices D^S and D'. This is optimized for $x \in \Theta(n^{2/3})$. In Section 4.1 we present Algorithm 6 and its subroutines and give the full proof of Theorem 2.2.

Theorem 2.2. *There is an algorithm that solves APSP in $\widetilde{O}(n^{2/3})$ rounds w.h.p.*

2.4 Upper Bounds for Approximate APSP. The bottleneck of the exact algorithm is the dissemination of h-hop limited distances between all pairs of nodes in $V \setminus M \times M$, which takes $\widetilde{O}(n/\sqrt{x})$ rounds. Our approximative approach (Algorithm 10) mitigates this bottleneck by disseminating only *one* distance token d_{uv} per $v \in V \setminus M$, representing the distance between v and its *closest* marked node $u \in M$, which can be done $\widetilde{O}(\sqrt{n})$ rounds. We show that for suitable choices of x, Algorithm 10 can be used to obtain a 3-approximation of APSP for general (connected) graphs in $\widetilde{O}(\sqrt{n})$ and a $(1+\varepsilon)$-approximation for unweighted graphs in $\widetilde{O}(\sqrt{n/\varepsilon})$.

Besides slightly adapted procedures, Algorithm 10 uses the same subroutines as in the exact case. After all subroutines are performed, each node in V knows (w.h.p.): (i) its m-limited distances $d_m(\cdot, v)$ to any other node $v \in V$ (ii) the skeleton S and the distance matrix D^S among its nodes (c.f., Lemma 4.2) and (iii) the distance $d_{vv'}$ between any node $v \in V \setminus M$ and its respective closest marked node $v' \in M$. With this knowledge, each node computes an approximative distance Matrix $\tilde{D}^G$ of G as follows:

$$\tilde{D}^G_{uv} = \min\Big(d_m(u,v), \min_{u' \in M}\big(d_h(u,u') + D^S_{u'v'}\big) + d_{vv'}\Big),$$

where $v' \in M$ is v's closest marked node and $m = \max\big(h, \frac{n}{h}\big)$. In Section 4.2 and [5] we show the following:

Theorem 2.3. *There is an algorithm to compute a 3-approximation of APSP in $\widetilde{O}(\sqrt{n})$ rounds w.h.p.*

THEOREM 2.4. *For any $\varepsilon > 0$, there is an algorithm that computes a $(1+\varepsilon)$-approximation of the APSP problem on unweighted graphs in $\widetilde{O}\big(\sqrt{n/\varepsilon}\,\big)$ rounds w.h.p.*

2.5 Lower Bounds for APSP. In order to obtain rigorous lower bounds we introduce the technical Lemma 4.4 in Section 4.3. It shows that for a class of graphs and a dedicated node b, we can create a bottleneck for the information that can be transmitted from parts of the graph to b. More specifically, we show that if the state of some random variable X is given to the nodes of some subgraph G' and if b is at the end of some path of length L, then every randomized algorithm in which b needs to learn the state of X requires $\widetilde{\Omega}\big(\min(L, H(X)/L)\big)$ rounds, where $H(X)$ denotes the Shannon entropy of X (c.f., Figure 1).

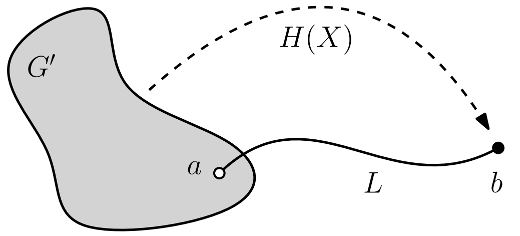

Figure 1: Construction of Lemma 4.4.

We use Lemma 4.4 to show a lower bound of $\widetilde{\Omega}\big(\sqrt{n}\big)$ for APSP that is robust even if we allow approximation factors up to some $\alpha = \widetilde{O}\big(\sqrt{n}\big)$ (naturally, we restrict ourselves to unweighted graphs, c.f., Theorem 2.5, full proof in Section 4.3). The idea is to construct an unweighted graph consisting of a path of length $\Omega(n)$ and two node sets S_1, S_2 of size $\Omega(n)$ each. Let b be on one end of the path. Then we attach two nodes sets S_1 and S_2 of size $\Omega(n)$ each to the path at distance $L = \widetilde{\Omega}(\sqrt{n})$ and $\Omega(n)$ from b (c.f., Figure 2).

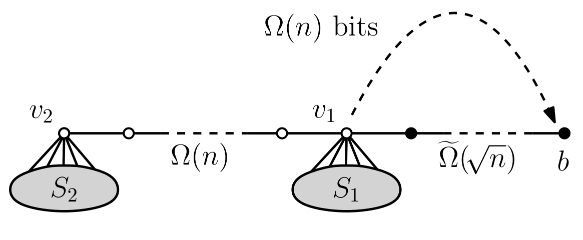

Figure 2: Construction of Theorem 2.5.

Now assume an adversary "shuffles" the nodes in S_1, S_2 uniformly at random, where the state of S_1 will be our random variable X. If b does not know for a node u whether $u \in S_1$ or $u \in S_2$, then b must assume $u \in S_2$ (recall that approximations must be lower bounded by the true distance). However, if $u \in S_1$ would be true, then this results in a larger approximation ratio than we allow. We show that in order to learn S_1, node b needs to receive $H(X) = \Omega(n)$ bits. Choosing $L = \widetilde{\Theta}\big(\sqrt{n}\big)$ yields the claimed lower bound.

THEOREM 2.5. *An α-approximative APSP algorithm in the hybrid network model on unweighted graphs takes $\Omega\big(\sqrt{n}/\log n\big)$ rounds, for any $\alpha \le \sqrt{nc} \cdot \log n/2$, where $c \log n$ (for constant $c > 0$) is the number of messages a node can receive per round over global edges.*

2.6 Upper Bounds For Exact SSSP. We give a sketch of the exact SSSP algorithm, whereas the formal algorithm and proofs can be found in Section 5.1. Throughout the algorithm, every node v maintains a distance estimate $\hat{d}(v)$, which is initialized to $\hat{d}(v) = \infty$, for $v \in V \setminus \{s\}$, and $\hat{d}(s) = 0$. Eventually, we will have that $\hat{d}(v) = d(s, v)$ for every node $v \in V$. The algorithm proceeds in phases $i = 1, \dots, \lceil 2\sqrt{\mathsf{SPD}}\,\rceil$. At the beginning of each phase i, two invariants are maintained: (i) every node $v \in V$ knows the subgraph $G(v, 2i)$ of G induced by all nodes within hop-distance $2i$ to v, and (ii) $\hat{d}(v) = d(s, v)$ for every $v \in V$ with $sph(s, v) \le t(i-1)$, where $t(i) := \sum_{j=1}^{i} j$ denotes the i-th triangular number. Since there exists a shortest path of at most SPD hops between any two nodes, and $t(\lceil 2\sqrt{\mathsf{SPD}}\,\rceil) \ge \mathsf{SPD}$, after $\lceil 2\sqrt{\mathsf{SPD}}\,\rceil$ phases every node knows its exact distance to s. We will later show that each phase only takes time $\widetilde{O}(1)$, which implies a runtime of $\widetilde{O}(\sqrt{SPD})$.

Maintaining Invariant (i) is simple: every node sends all information it has learned about the graph so far to its neighbors for two rounds via local edges at the beginning of each phase. Maintaining Invariant (ii) is the main concern of our algorithm. Note that from Invariant (i) every node v knows $d_{G(u,i)}(u, v)$ for every $u \in G(v, i)$. If $sph(s, v) \le t(i)$, and v would also know the distance estimate of every node $u \in G(v, i)$, then, since there must be a node on a shortest path from s to v that is within i hops from v and that knows its correct distance to s already, v could easily determine $d(s, v)$ with the equation

$$d(s, v) = \min_{u \in G(v,i)} \Big(\hat{d}(u) + d_{G(u,i)}(u, v)\Big). \tag{2.1}$$

Unfortunately, naively exchanging all distance estimates among all pairs of nodes within i hops of each other over the global network in order to compute Equation 2.1 would either take too long or cause too much contention on the global network, as the neighborhood of a node could be of size $\Theta(n)$. However, we will exploit the fact that it suffices that node v learns the distance label $d_{uv} := \hat{d}(u) + d_{G(u,i)}(u, v)$ from some node u that minimizes Equation 2.1 and can safely disregard distance labels of other nodes in $G(v, i)$.

We define $T(u,i)$ as the shortest-path tree of $G(u,i)$ rooted at u, in which the parent of every node v is chosen as its neighbor w in $G(u,i)$ that minimizes $d_{G(u,i)}(u,w)$; if there are multiple such neighbors, we choose the one that minimizes $sph_{G(u,i)}(u,w)$ (breaking ties by choosing the w with smallest identifier). Due to Invariant (i), u and any node $v \in G(u,i)$ know $T(u,i)$. The goal of u is to propagate the respective distance label d_{uv} to all nodes v in $T(u,i)$ for which u minimizes Equation 2.1. To achieve that, every node u initiates a recursive divide-and-conquer strategy, where the executions of all nodes are performed in parallel. First, u starts with the tree $T := T(u,i)$. In each recursion level, the tree T is split at a *splitting node* x whose removal decomposes T into subtrees of size at most $|T|/2$. As we show in Lemma 5.1, such a node always exists and can be computed locally by all $v \in G(u,i)$ (due to Invariant (i)).

Let T_x be the subtree of T rooted at x. The root u of T will take care of informing every node $v \in T \setminus T_x$ about d_{uv} in the next recursion, whereas the task of informing the nodes $v \in T_x$ about d_{uv} is delegated to the children of x. For that purpose, u informs x about the distance $d_{ux} := \hat{d}(u) + d_{G(u,i)}(u,x)$. Subsequently, x instructs every child c in T to start another recursion in their respective subtree T_c by sending it the distance $d_{uc} := d_{ux} + w(x,c)$ via the local edge $\{x,c\}$.

Note that x might have been chosen as a splitting by multiple roots; therefore, sending d_{ux} to x over the global network might cause too much contention. We carefully resolve this by making every root w of some tree T that intends to send a value d_{wx} to x participate in an *aggregation*[11] towards x using Theorem 2.2 of [4] as a black box. Each root w injects a message containing d_{wx}, $sph_{T(w,i)}(w,x)$ and $\mathrm{id}(w)$ into the aggregation process, whereas we aggregate for d_{wx} and break ties by the smallest hop value $sph_{T(w,i)}(w,x)$, and then by the smallest identifier. Note that in the aggregation process, the message for x injected by the root u might be blocked by some other message, which disrupts the recursion intended by u to be continued at x. However, as shown in Section 5.1, continuing only the most promising recursions is sufficient for x to obtain the distance label minimizing Equation 2.1.

In the subsequent recursion level, every node has to continue all recursions it handles so far, whereas the corresponding trees at least halve in size. Each node obtains at most one additional recursion from being the child of a splitting node in the last recursion level. Since the subtree of each recursion consists of a single node after $O(\log n)$ recursion levels, the number of recursions handled simultaneously by any node is $O(\log n)$. After all recursions have finished, every node v with $sph(s,v) \leq t(i)$ has $\hat{d}(v) = d(s,v)$, which maintains Invariant (ii) for the next iteration $i+1$. A formal proof of the following theorem can be found in Section 5.1.

[11]The aggregation protocol solves the following problem. Given an *aggregation function* (e.g. MIN, MAX, SUM) and a set of source nodes that hold inputs, then some set of target nodes has to learn the result of the function applied to a subset of inputs.

THEOREM 2.6. *There is an algorithm that solves SSSP in time $\widetilde{O}\big(\sqrt{\mathsf{SPD}}\big)$, w.h.p.*

2.7 Approximate SSSP in $\widetilde{O}(n^{1/3})$ Rounds.

At its core, our $\big(1+o(1)\big)$-approximate SSSP algorithm relies on simulating the algorithm of Becker et al. [7] that computes a $(1+\varepsilon)$-approximation of SSSP for the broadcast congested clique model (BCC) in $\widetilde{O}(1)$ rounds. First, we compute a skeleton $S = (M, E_S)$ with $|M| = \widetilde{\Theta}(n^{2/3})$ with edges between nodes at most $h = \widetilde{O}(n^{1/3})$ hops apart (note that we always include the source $s \in M$). Using token dissemination (c.f., Section 3), we can simulate one round of the BCC model on S in $\widetilde{O}(n^{1/3})$ time. This allows us to simulate the algorithm of [7] on S to solve $(1+\varepsilon)$-approximate SSSP on S in $\widetilde{O}(n^{1/3}/\varepsilon^6)$ rounds in our model.

By using token dissemination again, we make the distance approximations that we computed for pairs $M \times \{s\}$ publicly known in $\widetilde{O}(n^{1/3})$ rounds. In another $h = \widetilde{O}(n^{1/3})$ rounds, all nodes in V can do a local search to determine the distance to close nodes in M. After these steps, every node $v \in V$ knows approximate distances $\tilde{d}_{su}$ between s and any marked node $u \in M$ and also its own h-limited distance $d_h(u,v)$ to any marked node $u \in M$. Then, every node $v \in V$ locally computes an approximate distance $\tilde{d}_{sv}$ to s using the following equation:

$$\tilde{d}_{sv} := \min\Big(d_h(s,v), \min_{u \in M}\big(\tilde{d}_{su} + d_h(u,v)\big)\Big)$$

THEOREM 2.7. *There is an algorithm that $(1+\varepsilon)$-approximates SSSP in $\widetilde{O}(n^{1/3}/\varepsilon^6)$ rounds, w.h.p.*

Choosing, for example, $\varepsilon = \frac{1}{\log n}$ we obtain a $(1+o(1))$-approximate SSSP algorithm with complexity $\widetilde{O}(n^{1/3})$. More details on the algorithm and the proof of Theorem 2.7 are given in Section 5.2.

2.8 Approximate SSSP in $\widetilde{O}(n^{\varepsilon})$ Rounds.

Finally, we present a $(\log_\alpha n)^{O(\log_\alpha n)}$-approximate SSSP algorithm that takes time $\widetilde{O}(\alpha^3)$, w.h.p., for a parameter $\alpha \geq 5$. By setting $\alpha = n^{\varepsilon}$ for some $\varepsilon > 0$, we obtain a $(1/\varepsilon)^{O(1/\varepsilon)}$-approximation in time $O(n^{3\varepsilon})$, which, for example, allows to compute a constant factor approximation for any constant ε. Furthermore, for $\varepsilon =$

$\sqrt{\log\log n/\log n}$, this gives a $2^{O(\sqrt{\log n \log\log n})}$ approximate solution in subpolynomial time $2^{O(\sqrt{\log n \log\log n})}$. We describe the algorithm from a high level and provide all details in Section 5.3.

The main idea of the algorithm is to recursively construct a hierarchy of spanners $G_1, \dots, G_T$, where G_i is a spanner of the nodes in $M_i \subseteq V[G_{i-1}]$. The set M_i contains each node of G_{i-1} with probability $\log(n)/\alpha$ for $i = 2$, and with probability $1/\alpha$ for $i \geq 3$. The first spanner G_1, which contains all nodes of G, is constructed using only the local network by simply performing the distributed Baswana-Sen algorithm [6] with parameter k as a black box, which gives a $(2k-1)$-spanner in time $O(k^2)$.

The construction of subsequent spanners $G_2, \dots, G_T$ relies entirely on the global network. For $i \geq 2$, we construct G_i as an *h-hop skeleton spanner* of G_{i-1}.[12] Intuitively, a skeleton spanner gives a good approximation of distances between nodes that are within hop-distance h. Furthermore, the skeleton spanner has low arboricity. This allows us to ensure that every edge of the spanner G_i is learned by one endpoint of the edge in such a way that no node has to take care of more than $\widetilde{O}(\alpha)$ edges. On this graph, we can efficiently apply the techniques of [4] using the global network. More specifically, we will prove that we can construct G_i as $O(\alpha)$-hop skeleton spanner of stretch $O(\log_\alpha n)$ of the graph G_{i-1} for all $i \geq 2$.

Finally, by taking the union of all the graphs G_i, we obtain a global spanner H for the whole graph. Applying the properties of the skeleton spanners G_i, we show that H has a $(\log_\alpha n)^{O(\log_\alpha n)}$-approximate path consisting of at most $\widetilde{O}(\alpha)$ hops for every pair of nodes $u, v \in V$. Therefore, every node v learns a good approximation $\tilde{d}(s,v)$ of $d(s,v)$ by performing a distributed Bellman-Ford algorithm with source s in H for $\widetilde{O}(\alpha)$ rounds. Again using techniques of [4], one iteration of this algorithm can be realized in the global network in time $\widetilde{O}(\alpha)$. The following theorem results from a careful analysis of the approximation guarantees and runtime of our recursive spanner construction.

THEOREM 2.8. *There is an algorithm to compute a $(\log_\alpha n)^{O(\log_\alpha n)}$-approximation of SSSP in $\widetilde{O}(\alpha^3)$ rounds, w.h.p.*

3 Token Dissemination

In this section we give the details of Algorithm 1 solving the (k,ℓ)-TD problem and its subroutines. Finally, we provide a full proof of Theorem 2.1.

[12]Note that we do not compute a spanner of G_{i-1}, but of a *skeleton* of G_{i-1} (see Definition 1)

Algorithm 1 `Token-Dissemination` ▷ $x \in [2..k]$

```
Token-Balancing              ▷ redistribute s.t. $\ell = \widetilde{O}(\lceil \frac{k}{n} \rceil)$
Token-Multiplication         ▷ spread $\widetilde{O}(\frac{n}{k})$ copies
Token-Seeding($x$)           ▷ seed tokens with prob. $1/x$
Local-Dissemination($x$)     ▷ "collect" seeds
```

We start with a technical lemma showing that if every node sends $O(\log n)$ messages to random nodes, then every node receives only $O(\log n)$ messages w.h.p.

LEMMA 3.1. *Presume some algorithm takes at most $p(n)$ rounds for some polynomial p. Presume that each round, every node sends at most $\sigma = \Theta(\log n)$ messages via global edges to σ targets in V picked independently and uniformly at random. Then there is a $\rho = \Theta(\log n)$ such that for sufficiently large n, in every round, every node in V receives at most ρ messages per round w.h.p.*

The proof is a simple application of a Chernoff bound and a union bound. For completeness we provide it in the full version of this paper [5].

The following algorithm balances the number of tokens per node to $\widetilde{O}(\lceil k/n \rceil)$.[13] The set of tokens received during the execution of Algorithm 2 forms the new set of tokens a node has to take care of.[14]

Algorithm 2 `Token-Balancing` ▷ $\sigma = \Theta(\log n)$

```
$T_v \leftarrow$ initial set of tokens of this node $v$
for $O(\ell/\log n)$ rounds do                    ▷ redistribute tokens
    $U_v \leftarrow$ sample $\sigma$ nodes from $V$ i.i.d.  ▷ $U_v$ is multiset
    $S \leftarrow$ select subset of size $\min(\sigma, |T_v|)$ from $T_v$
    $v$ sends $S$ via global network to $U_v$ (one-to-one)
    $T_v \leftarrow T_v \setminus S$
```

LEMMA 3.2. *If each node holds at most ℓ tokens and there are k tokens in total, then Algorithm 2 redistributes all tokens in $O(\ell/\log n)$ rounds such that afterwards each node holds $O(\lceil \frac{k}{n} \rceil \log n)$ tokens w.h.p.*

Again the proof is a straight-forward application of a Chernoff bound and a union bound. We provide it in the full version [5] for completeness.

If k is small, the next algorithm boosts the number of nodes that hold a fixed token by a factor $\widetilde{\Theta}(\frac{n}{k})$.

[13]The term $\lceil k/n \rceil$ is shorthand for $\max(1, k/n)$ meaning that for $k \ll n$ we guarantee $\widetilde{O}(1)$ tokens per node.

[14]We write i.i.d. (independently, identically distributed) if we pick elements from some set uniformly at random.

Algorithm 3 `Token-Multiplication`

```
$T_v \leftarrow$ initial set of tokens of $v$          ▷ $T_v$ is a multiset
for $\lfloor \log_2(\frac{n}{k}) \rfloor$ phases do          ▷ runs only for $k \leq n/2$
    for each $t \in T_v$ do          ▷ $|T_v| = O(\log n)$ w.h.p.
        pick $u_1, u_2 \in V$ i.i.d.
        $v$ sends a copy of $t$ to $u_1, u_2$ via global network
    $T_v \leftarrow$ multiset of token-copies received this phase
```

LEMMA 3.3. *Presume that $k \leq n/2$ and each node has at most $O(\log n)$ tokens. By invoking Algorithm 3, each token t is copied to a random subset $V_t \subseteq V$ with $|V_t| \geq \frac{n}{k\zeta \ln n}$ w.h.p., for some constant $\zeta > 0$. Algorithm 3 takes $O(\log n \log \frac{n}{k})$ rounds. Afterwards, we still have $O(\log n)$ tokens per node.*

Proof. Note that if node v picks itself as recipient for a token copy (which we allow), it just keeps one for the next phase. Since we choose targets randomly, no node receives more than $O(\log n)$ messages per round w.h.p., due to Lemma 3.1.

Let $\varphi := \lfloor \log_2(\frac{n}{k}) \rfloor$ be the number of phases of Algorithm 3. In each phase the total number of copies of a token t in the whole network (stored locally in the variables T_v, $v \in V$) exactly doubles, even if multiple copies of t end up at the same node, since T_v is defined as a multiset. Notice that we only carry the token-copies received in the current phase over to the next phase. This means that the number of distinct nodes that hold a copy of t after φ phases is upper bounded by 2^φ, hence $|V_t| \leq 2^\varphi \leq \frac{n}{k}$.

For the lower bound of V_t we show that in every phase, the multiset T_v contains at most $O(\log n)$ copies of tokens w.h.p. We emphasize that for the sake of this proof we distinguish *token-copies* when counting them, even if they originate from the same token. Initially the claim is true due to the presumption. After that, Algorithm 3 distributes all created token-copies uniformly at random.

In every phase, given *one* specific copy of a token t, a given node receives that copy with probability exactly $\frac{1}{n}$. Hence, the expected number of copies of any token t is at most $\mathbb{E}(|T_v|) \leq k \cdot |V_t| \cdot \frac{1}{n} \leq 1$ (recall $|V_t| \leq \frac{n}{k}$). For an arbitrary constant $c > 0$ we obtain $\mathbb{P}\big(|T_v| > 1+3c\log n\big) \leq \frac{1}{n^c}$ with a Chernoff bound (Lemma A.1), i.e. $|T_v| = O(\log n)$. With a union bound (Lemma A.2), $|T_v| = O(\log n)$ holds w.h.p. for all nodes in all rounds of Algorithm 3. Since $|T_v|$ is also the time complexity of a single phase (c.f., Algorithm 3), this proves the running time of Algorithm 3.

Let constant $\zeta > 0$ be such that $|T_v| \leq \frac{\zeta}{2} \ln n$ for almost all n. This means any node holds at most $\frac{\zeta}{2} \ln n$ token-copies, when we treat copies of the same token t as *different* copies (T_v is a multiset). Conversely, the size of the *set* V_t represents the overall number of copies of a token t in the network, in case we count multiple copies of the same token t at the same node as *one*. Therefore, the upper bound of 2^φ for $|V_t|$ differs from the lower bound by a factor of at most $\frac{\zeta}{2} \ln n$. We obtain

$$|V_t| \geq \frac{2^\varphi}{\max_{v \in V} |T_v|} \geq \frac{2^{\varphi+1}}{\zeta \ln n} \geq \frac{2^{\log_2(n/k)}}{\zeta \ln n} = \frac{n}{k\zeta \ln n}.$$

We already established the fact that the number of tokens-copies per node is at most $|T_v| = O(\log n)$, thus the same is obviously true for the number of distinct tokens per node and we have $O(\log n)$ tokens per node w.h.p., after the execution of Algorithm 3. ◻

The goal of the next algorithm is to seed each token to roughly $\frac{n}{x}$ random nodes. For $k = \widetilde{\Omega}(n)$ we can afford to sample targets with probability $\frac{1}{x}$ for each token and send them via the global network within our target runtime (which is $O\big(\ell \cdot \min(k,n)/x\big)$). For small k we decrease the sampling rate to $\widetilde{O}\big(\frac{k}{nx}\big)$. The algorithm still works because now $|V_t| = \widetilde{\Theta}(\frac{n}{k})$ nodes are helping to seed token t (c.f., Lemma 3.3).

Algorithm 4 `Token-Seeding`(x)
▷ $x \in [2..k]$, $\sigma = \Theta(\log n)$, $\zeta > 0$ *is from Lemma 3.3*

```
$T_v \leftarrow$ initial set of tokens of this node $v$
for $t \in T_v$ do
    if $k \geq \frac{n}{2\zeta \ln n}$ then          ▷ full sampling rate
        $S_t \leftarrow$ sample from $V$ with probability $\frac{1}{x}$
    else          ▷ reduced sampling rate
        $S_t \leftarrow$ sample from $V$ with prob. $\frac{k}{nx} \cdot 2\zeta \ln n$
    while $S_t \neq \emptyset$ do          ▷ seeding token $t$
        $S \leftarrow$ pick i.i.d. subset of $S_t$ of size $\min(\sigma, |S_t|)$
        $v$ sends $t$ via global network to all $u \in S \setminus \{v\}$
        $S_t \leftarrow S_t \setminus S$
```

LEMMA 3.4. *If each node has initially at most ℓ tokens, then w.h.p. after Algorithms 3 and 4, each node knows any given token with probability at least $1/x$. Algorithm 4 takes $O\big(\ell \cdot \min(k,n)/x\big)$ rounds.*

Proof. Each node sends a token to at most σ *uniformly random* nodes (uniformly, since a priori, the probability of being selected as a target by a fixed node in some fixed round is equal for every node). As before we invoke Lemma 3.1 to argue that w.h.p., no node v receives more than $O(\log n)$ messages per round.

Next we show that the sampled sets S_t are not too large. First consider the case $k \geq \frac{n}{2\zeta \ln n}$. In this case we sample with probability $1/x$ thus $\mathbb{E}(|S_t|) = \frac{n}{x}$ and with a standard Chernoff bound we have $|S_t| = O(n \log n/x)$ w.h.p. Now consider $k < \frac{n}{2\zeta \ln n}$. In this case the sample probability is reduced to $\frac{2k\zeta \ln n}{xn}$. For the expectation we get $\mathbb{E}(|S_t|) = \frac{2k\zeta \ln n}{x}$. For some $c > 0$ and with a Chernoff bound we obtain

$$\mathbb{P}\Big(|S_t| > \Big(1+\frac{3c}{2\zeta}\Big)\cdot\frac{2k\zeta \ln n}{x}\Big) \le \exp\Big(-\frac{kc\ln n}{x}\Big) \overset{x\le k}{\le} \frac{1}{n^c},$$

thus the event $|S_t| = O\big(k \log n/x\big)$ occurs w.h.p. Combining both cases we have $|S_t| = O\big(\min(k,n)\log n/x\big)$ w.h.p. In accordance with Lemma A.2 this is true for every node $v \in V$, every token $t \in T_v$ and in every round of Algorithm 4. Now we are able to compute the time complexity: Sending every token $t \in T_v$ to each of the sampled nodes in S_t takes $O\big(|T_v|\cdot|S_t|/\sigma\big) = O\big(\ell \cdot \min(k,n)/x\big)$ rounds.

It remains to be shown that in case $k < \frac{n}{2\zeta \ln n}$ we can still guarantee that each node obtains a given token with probability at least $1/x$ even though we sample with reduced probability $\frac{2k\zeta \ln n}{xn}$. In that case, we know from Lemma 3.3 that during a run of Algorithm 3 each token t is copied to a random subset $V_t \subseteq V$ of nodes of size $|V_t| \ge \frac{n}{k\zeta \ln n}$. In Algorithm 4 all nodes of V_t take part in seeding t. Let $p := \frac{2k\zeta \ln n}{xn}$ and let $q := \frac{n}{k\zeta \ln n}$. The probability that a fixed node receives a fixed token is at least $1-(1-p)^q$. We will show that $1-(1-p)^q \ge \frac{pq}{2} = \frac{1}{x}$ for $0 \le pq \le 1$ (the latter holds for $x \ge 2$). We have

$$(1-p)^q = \Big(\big(1-\frac{1}{1/p}\big)^{1/p}\Big)^{pq} \le e^{-pq} \le 1-\frac{pq}{2}.$$

The last inequality holds since for $pq = 0$ we have equality, for $pq = 1$ we have $e^{-1} < \frac{1}{2}$ and since e^{-x} is convex it holds for all values in between. □

```
Algorithm 5 Local-Dissemination($x$)          ▷ $x \in [2..k]$
  $T_v \leftarrow$ all tokens that $v$ learned during Token-Seeding
  for $r = O(x \log n)$ rounds do
      $v$ sends $T_v$ to all its neighbors in $G$ via local edges
      $T_v \leftarrow$ tokens $v$ learned for the first time last round
```

The proof of the following lemma is another straight-forward application of Chernoff and union bounds. We provide it in [5].

LEMMA 3.5. *Let G be a connected graph. If for all tokens t, any given node knows t with probability at least $\frac{1}{x}$, then after Algorithm 5 is performed, all nodes know all tokens w.h.p. after $O(x \log n)$ rounds.*

We stitch together the above results to prove Theorem 2.1 (i.e., Algorithm 1 solves (k,ℓ)-TD in $\widetilde{O}\big(\sqrt{k}+\ell\big)$).

Proof. [Proof of Theorem 2.1] Let ℓ_{init} be the initial maximum number of tokens per node. First we execute Algorithm 2: Token-Balancing which takes $O(\ell_{\text{init}})$ rounds. Afterwards we have $\ell = O\big(\lceil \frac{k}{n}\rceil \log n\big)$ tokens per node w.h.p., in accordance with Lemma 3.2.

In case $k \le n/2$ we run at least one phase of Algorithm 3: Token-Multiplication. Due to $k \le n/2$ we have $\ell = O(\log n)$ and therefore Algorithm 3 takes $\widetilde{O}(1)$ rounds according to Lemma 3.3. We have shown that the condition $\ell = O(\log n)$ is preserved by Algorithm 3.

In accordance with Lemma 3.4, Algorithm 4: Token-Seeding takes $O\big(\ell \cdot \min(k,n)/x\big)$ rounds. The maximum number of tokens is $\ell = O\big(\lceil \frac{k}{n}\rceil \log n\big)$. Since $O\big(\min(k,n)\lceil \frac{k}{n}\rceil\big) = O(k)$ the time complexity of Algorithm 4 reduces to $\widetilde{O}\big(\frac{k}{x}\big)$.

After Token-Seeding has terminated, the premise of Lemma 3.5 is fulfilled. Thus Algorithm 5: Local-Dissemination solves the (k,ℓ)-TD Problem in $\widetilde{O}(x)$ rounds.

The total number of rounds that Algorithm 1: Token-Dissemination takes is $\widetilde{O}(x) + \widetilde{O}\big(\frac{k}{x}\big) + O(\ell_{\text{init}})$. This is optimized for $x = \Theta\big(\sqrt{k}\big)$, which results in the overall time complexity $\widetilde{O}\big(\sqrt{k} + \ell_{\text{init}}\big)$. □

The subsequent lemma shows the extent to which we utilize the local network. The proof is given in the full version [5] of this paper.

LEMMA 3.6. *Algorithm 1 works in the same time $\widetilde{O}\big(\sqrt{k}+\ell\big)$ for local capacity $\lambda = \Theta\big(\sqrt{k}\big)$.*

4 All Pairs Shortest Paths

This section focuses on the APSP problem. We first show how to solve APSP exactly. Second, we show that a significant improvement in the time complexity is possible, if we restrict ourselves to approximations. Finally, we prove that the running times of our approximate algorithms are tight up to polylog n factors.

4.1 Upper Bounds for Exact APSP.

In the following we present Algorithm 6 and its subroutines and show their properties. Subsequently we prove Theorem 2.2.

```
Algorithm 6 Exact-APSP                              ▷ $x \in [1..n]$
  Construct-Skeleton($x$)     ▷ construct $S = (M, E_S)$
  Transmit-Skeleton           ▷ learn dist. of pairs $M \times M$
  Transmit-Distances          ▷ learn dist. of pairs $V \times M$
  compute APSP distances locally with Equation (4.2)
```

(4.2)
$$D^G_{uv} = \min\Big(d_h(u,v), \min_{u',v' \in M}\big(D'_{uu'} + D^S_{u'v'} + D'_{vv'}\big)\Big).$$

First we construct the skeleton S by sampling its nodes and then determining its edges via exploration of the local network. As byproduct of the latter, all nodes learn their h-hop neighborhood in G.

Algorithm 7 `Construct-Skeleton`(x) $\quad\triangleright\ x \in [1..n], h := \xi x \ln n$

```
v is marked with prob. $\frac{1}{x}$          ▷ marked nodes form $M$
for $h$ rounds do                   ▷ initially $v$ sends incident edges
    $v$ sends edges it learned last round to its neighbors
```

FACT 4.1. *The size of M is $\widetilde{O}(n/x)$ w.h.p. Let $h := \xi x \ln n$. Algorithm 7 establishes a weighted graph $S = (M, E_S)$ among the set of marked nodes M in $O(x \log n)$ rounds, whereas we define $E_S := \{\{u,v\} \mid u,v \in M, hop(u,v) \le h\}$. The weight of $\{u,v\} \in E_S$ is defined as $d_h(u,v)$. After the subroutine, all nodes $v \in V$ know all $u \in V$ that are within h hops as well as the distances $d_h(u,v)$. Specifically, this means that all marked nodes know their neighbors in S and the distances of the incident edges in E_S.*

The following lemma shows that for nodes at sufficient hop-distance $\widetilde{\Omega}(h)$, there is a marked node every $\widetilde{O}(h)$ hops on some shortest path between those nodes.

LEMMA 4.1. *Let M be a subset of V created by marking each of the nodes of V with probability at least $\frac{1}{x}$. Then there is a constant $\xi > 0$, such that for any $u,v \in V$ with $hop(u,v) \ge \xi x \ln n$, there is at least one shortest path P from u to v, such that any sub-path Q of P with at least $\xi x \ln n$ nodes contains a node in M w.h.p.*

Proof. Let $u,v \in V$ with $hop(u,v) \ge \xi x \ln n$. Fix a shortest u-v-path $P_{u,v}$ and let Q be a sub-path of $P_{u,v}$ with at least $\xi x \ln n$ nodes. Let $X_{u,v}$ be the random number of marked nodes on Q. Then we have $\mathbb{E}(X_{u,v}) \ge \frac{|Q|}{x} \ge \xi \ln n$. Let $c > 0$ be arbitrary. We use a Chernoff bound:

$$\mathbb{P}\Big(X_{u,v} < \frac{\xi \ln n}{2}\Big) \le \exp\Big(-\frac{\xi \ln n}{8}\Big) \overset{\xi \ge 8c}{\le} \frac{1}{n^c}.$$

Thus we have $X_{u,v} \ge 1$ w.h.p. for constant $\xi \ge \max(8c, 2/\ln n)$. Therefore the claim holds w.h.p. for the pair u,v. We claim that w.h.p. the event $X_{u,v} \ge 1$ occurs for all pairs $u,v \in V$ and for all sub-paths Q of $P_{u,v}$ longer than $\xi x \ln n$ hops, for at least one shortest path $P_{u,v}$ from u to v. There are at most n^2 many pairs $u,v \in V$. Moreover we can select at most n sub-paths Q of P that do not fully contain any other selected sub-path. Hence the claim follows with the union bound given in Lemma A.2. ∎

Next, we make the skeleton publicly known via token dissemination.

Algorithm 8 `Transmit-Skeleton` $\quad\triangleright\ h = \xi x \ln n$

```
if $v$ is marked then
    for each $\{u,v\} \in E_S$ do        ▷ local computation
        create token $t_{u,v} = \langle ID(u), ID(v), d_h(u,v)\rangle$
Token-Dissemination          ▷ dissem. $E_S$ and weights
```

LEMMA 4.2. *After Algorithm 8, w.h.p. every node knows the skeleton S and has sufficient information to locally compute a distance matrix D^S of S, with $D^S_{uv} = D^G_{uv}$ for all $u,v \in M$ (where D^G denotes the true distance matrix of G), if $h = \xi x \ln n$ for appropriately chosen constant ξ. Algorithm 8 takes $\widetilde{O}\big(\frac{n}{x}\big)$ rounds.*

Proof. First we point out that every marked node $v \in M$ knows $d_h(u,v)$ for all $u \in V$ due to Fact 4.1 (recall that we set $d_h(u,v) := \infty$ if $hop(u,v) > h$) and is thus able to create the tokens described in the algorithm. Each marked node creates at most $\ell = |M|$ tokens of size $O(\log n)$ (recall that weights are polynomially bounded in n). The total number of created tokens is at most $k = |M|^2$. By Theorem 2.1, Algorithm 8 takes $\widetilde{O}\big(|M|\big) = \widetilde{O}(n/x)$ rounds. After the token dissemination *every* node knows *every* edge $\{u,v\} \in E_S$ as well as its weight, defined as h-limited distance $d_h(u,v)$ (in Fact 4.1).

Let $h := \xi x \ln n$ (where ξ is the constant from Lemma 4.1) and let $u,v \in M$. If there is a shortest u-v-path P with $|P| \le h$, then obviously the weight $d_h(u,v)$ of the skeleton edge $\{u,v\} \in E_S$ equals D^G_{uv} (let us denote this fact with (i)). Otherwise $|P| > h$ for any shortest u-v path P. Then Lemma 4.1 implies that w.h.p., within every h hops of P there must be at least one marked node (we denote this fact with (ii)). This entails that S is connected if G is connected; and G is connected by definition (let this fact be (iii)).

From (i),(ii) and (iii) we deduce that every node can compute D^S by locally solving APSP on S. ∎

It remains to transmit the distances between skeleton nodes and non-skeleton nodes.

Algorithm 9 `Transmit-Distances` $\quad\triangleright\ h = \xi x \ln n$

```
if $v$ not marked then
    for each $u \in M$ do            ▷ local computation
        create token $t_{u,v} = \langle \mathrm{id}(u), \mathrm{id}(v), d_h(u,v)\rangle$
Token-Dissemination   ▷ dissem. dist. of $V\setminus M \times M$
```

FACT 4.2. *Algorithm 9 disseminates $d_h(u,v)$ for all $u \in M$ and $v \in V\setminus M$ to all nodes in the network (recall that we define $d_h(u,v) := \infty$ if $hop(u,v) > h$). The h-limited distances $d_h(u,v)$ are known to v due to Fact 4.1. Each node creates at most $\ell = |M|$ tokens (of size $O(\log n)$*

bits), thus there are at most $k = n|M|$ tokens in total. Due to Theorem 2.1, Algorithm 9 takes $\widetilde{O}\big(\sqrt{n|M|}\big)$ rounds. Since $|M| \in \widetilde{O}\big(\frac{n}{x}\big)$ w.h.p., this translates into a running time of $\widetilde{O}\big(n/\sqrt{x}\big)$ rounds.

We put everything together and show that Algorithm 6 solves the APSP problem in $\widetilde{O}\big(n^{2/3}\big)$ rounds.

Proof. [Proof of Theorem 2.2] After the first two subroutines of Algorithm 7, due to Lemma 4.2, every node knows S and can locally compute the distance Matrix D^S among all nodes in M. Additionally, based on the h-limited distances disseminated by Algorithm 9 as described in Fact 4.2, every node can locally compute the matrix

$$D' := \big(d_h(u,v)\big)_{u \in V \setminus M, v \in M}.$$

Let $u, v \in V$. If there exists a shortest u-v-path that has at most h hops, then $d(u,v) = d_h(u,v)$ which both u and v already know due to the local exploration conducted in Algorithm 7 (c.f. Fact 4.1). Otherwise, we infer from Lemma 4.1 that w.h.p., there is shortest u-v-path P with two marked nodes $u', v' \in M$ with $hop(u,u'), hop(v,v') \leq h$ (possibly $u' = v'$). We have

$$\begin{aligned} d(u,v) &= d_h(u,u') + d(u',v') + d_h(v',v) \\ &= D'_{uu'} + D^S_{u'v'} + D'_{vv'}. \end{aligned}$$

Hence every node can locally compute the complete distance matrix D^G of G as follows (we set $D'_{uv} = 0$ if $u, v \in M$):

$$D^G_{uv} = \min\Big(d_h(u,v), \min_{u',v' \in M}\big(D'_{uu'} + D^S_{u'v'} + D'_{vv'}\big)\Big).$$

The total running time is $\widetilde{O}(x) + \widetilde{O}\big(n/\sqrt{x}\big)$ due to Fact 4.1, Lemma 4.2 and Fact 4.2. This is optimized for $x \in \Theta\big(n^{2/3}\big)$. ◻

Finally, we show that local capacity $\lambda = \Theta(n)$ suffices to solve APSP exactly in the claimed time. We give a full account of the proof in [5].

LEMMA 4.3. *Algorithm 6 works in the same time $\widetilde{O}(n^{2/3})$ for local capacity $\lambda = \Theta(n)$.*

4.2 Upper Bounds for Approximate APSP. Besides slightly adapted procedures `Transmit-Closest` and `Construct-Skeleton'`, Algorithm 10 uses the same subroutines as Algorithm 6 to construct and disseminate the skeleton S and then determine its edges with a local search via the physical edges. In the following we briefly explain the (minor) changes of the subroutines of Algorithm 10.

Algorithm 10 `Approximative-APSP` ▷ $x \in [1..n]$

```
Construct-Skeleton'($x$)   ▷ construct $S = (M, E_S)$
Transmit-Skeleton          ▷ learn dist. of pairs $M \times M$
Transmit-Closest           ▷ learn dist. to close nodes in $S$
approximate APSP distances with Equation (4.3)
```

(4.3)
$$\tilde{D}^G_{uv} = \min\Big(d_m(u,v), \min_{u' \in M}\big(d_h(u,u') + D^S_{u'v'}\big) + d_{vv'}\Big).$$

As a slight adaption over the exact variant, we conduct a local exploration (Algorithm 11) up to hop-distance $m = \max\big(h, \frac{n}{h}\big)$ (instead of h), which allows us to use the same algorithm to compute approximations for the weighted case and the unweighted case (with different parameters x). The complexity of $O(m) = \widetilde{O}\big(\max(n, \frac{n}{x})\big)$ of `Approximative-APSP` does not change the analysis, as both factors $\widetilde{O}(x)$ and $\widetilde{O}(\frac{n}{x})$ appear in the analysis, regardless.

Algorithm 11 `Construct-Skeleton'`(x) ▷ $x \in [1..n], h := \xi x \ln n$

```
$v$ is marked with prob. $\frac{1}{x}$   ▷ marked nodes form $M$
for $m = \max\big(h, \frac{n}{h}\big)$ rounds do
    $v$ sends edges it learned last round to its neighbors
```

As to the main change, instead of disseminating distances of all pairs $V \times M$ as we did in in Algorithm 9, the dissemination is restricted to distances from each node to its closest skeleton node in Algorithm 12. With this modification we can still convey enough distance information to all nodes, such that they can compute meaningful approximations, while significantly decreasing the amount of data that has to be disseminated.

Algorithm 12 `Transmit-Closest` ▷ $h = \xi x \ln n$

```
if $v$ not marked then
    $v' \leftarrow \arg\min_{w \in M} d_h(v,w)$   ▷ $v' \in M$ closest to $v$
    $d_{vv'} \leftarrow d_h(v,v')$   ▷ dist. from $v$ to closest $v' \in M$
    create token $t_{v',v} = \langle \mathrm{id}(v), \mathrm{id}(v'), d_{vv'} \rangle$
Token-Dissemination   ▷ disseminate distances $d_{vv'}$
```

We prove Theorems 2.3 and 2.4 in the full version of this paper [5]. The proofs mainly rely on applications of the triangle inequality to show the claimed approximation ratios of 3 for the weighted and $(1+\varepsilon)$ for the unweighted case. Proving the properties of the skeleton and the running times of $\widetilde{O}\big(\sqrt{n}\big)$ rounds for the weighted and $\widetilde{O}\big(\sqrt{n/\varepsilon}\big)$ rounds for the unweighted case essentially follows the script of Section 4.1.

4.3 Lower Bounds for APSP.

We give a technical lemma that shows that for a class of graphs and a dedicated node b, the information per round that can be transmitted from parts of the graph to b is inherently limited. We prove this lemma in the full paper [5]. Subsequently we show that obtaining solutions (or even approximations) for the all pair shortest path problem requires that a certain amount of information (measured in terms of its entropy) must be transmitted to b, which demonstrates the lower bounds claimed in this section.

LEMMA 4.4. *Let $G = (V, E)$ be an n-node graph that consists of a subgraph $G' = (V', E')$ and a path of length L (edges) from some node $a \in V'$ to $b \in V \setminus V'$ and that except for node a is vertex-disjoint from V'. Assume further that the nodes in V' are collectively given the state of some random variable X and that node b needs to learn the state of X. Every randomized algorithm that solves this problem in the hybrid network model requires $\Omega\Big(\min\big(L, \frac{H(X)}{L\cdot\log^2 n}\big)\Big)$ rounds, where $H(X)$ denotes the Shannon entropy of X.*

Proof. [Proof of Theorem 2.5] We construct an unweighted graph G in which an α-approximative APSP algorithm has the claimed lower bound (c.f., Theorem 2.5). One part of G is a path with x nodes, where $x := \lfloor n/2 + L\rfloor$ and $L := \big\lfloor \frac{\sqrt{n}}{\sqrt{c}\log n}\big\rfloor$. Node b is at an end of the path. Moreover, G has two sets of nodes S_1, S_2, of size $y := \big\lfloor \frac{n-x}{2}\big\rfloor$.

We have $x, y \in \Omega(n)$. Note that we round x, y such that G has $x+2y \in [(n-3)..n]$ nodes in total. This is w.l.o.g. since we can always attach add a few additional nodes to the path. Every node in S_1 has an edge to v_1 which is the node with $hop(v_1, b) = L$. Every node in S_2 has an edge to v_2, which is the node with $hop(v_2, b) = x$.

We allow that the nodes that are on the path from v_2 to b are fixed and globally known. However, we assign the $2y$ remaining nodes randomly to S_1 and S_2. Formally, we fix a set of y nodes $u_1, \dots, u_y$ and assign each randomly to S_1 or S_2 with probability 1/2. The last y nodes are used to fill up S_1 and S_2 to size y.

Let $\mathcal{A}$ be an algorithm that computes an α-approximation of APSP for $\alpha \le \sqrt{nc}\cdot\log n/2$. In order to approximate APSP, node b needs to determine a distance estimation $\tilde{d}(b, u)$ for each $u \in \{u_1, \dots, u_y\}$ such that $d(b, u) \le \tilde{d}(b, u) \le \alpha \cdot d(b, u)$. If b does not know whether $u_i \in S_1$ or $u_i \in S_2$ for one node u_i with $i \in [1..y]$, then the best, valid estimation b can make is $\tilde{d}(b, u_i) = x$ under the assumption that $u_i \in S_2$. If however $u_i \in S_1$ is true, then the approximation ratio α' would be

$$\alpha' = \frac{\tilde{d}(b, u_i)}{d(b, u_i)} = \frac{x}{L} = \frac{\lfloor \frac{n}{2} + L\rfloor}{L} > \frac{n}{2L} > \frac{\sqrt{nc}\log n}{2} \ge \alpha.$$

Hence b must learn whether $u_i \in S_1$ or $u_i \in S_2$ for all $i \in [1..y]$. Let $X \in \{1, 2\}^y$ be the random assignment $u_1, \dots, u_y$ either to S_1 or S_2. I.e., X represents the outcome of a y-fold Bernoulli process. Since each outcome $o \in \{1, 2\}^y$ is equally probable, we have $p := \mathbb{P}(X\!=\!o) = 1/|X| = 1/2^y$. Thus the entropy of X is

$$H(X) = \sum_{o\in\{1,2\}^y} p\log(1/p) = 2^y \frac{1}{2^y}\log(2^y) = y \in \Omega(n)$$

Now the conditions of Lemma 4.4 apply, hence it takes at least $\Omega\big(\min(L, \frac{n}{L\cdot\log^2 n})\big) = \Omega\big(\sqrt{n}/\log n\big)$ rounds until b knows the state of X, which is - as we showed - a requirement in order that $\mathcal{A}$ can compute an α-approximation for APSP on G. $\square$

5 Single-Source Shortest Paths

The final section revolves around computing single-source shortest path distances. We first present an exact algorithm to solve SSSP in time $\widetilde{O}\big(\sqrt{\mathsf{SPD}}\big)$. Subsequently, we give two algorithms that approximate SSSP for various running times and approximation factors.

5.1 Exact SSSP.

The first part of this section is devoted to a detailed and formal description of our exact SSSP algorithm. The pseudocode of the algorithm can be found in [5]. For a more intuitive description of the algorithm we refer the reader to Section 2.6. Subsequently, we prove important properties of the algorithm, from which we can infer Theorem 2.6.

As described before, the algorithm proceeds in phases $i = 1, \dots, \lceil 2\sqrt{SPD}\rceil$, where, roughly speaking, the goal of each node v in phase i is to propagate distance information to s to all nodes within hop-distance i. Recall that at the beginning of phase i, Invariant (i) states that v knows the subgraph $G(v, 2i)$ of G induced by all nodes within hop-distance $2i$, and Invariant (ii) ensures that the distance estimate $\hat{d}(v) = d(u, v)$ for every node $v \in V$ with $sph(s, v) \le t(i-1)$, where $t(i) := \sum_{j=1}^{i} j$.

To ensure Invariant (i), at the beginning of phase i, v first sends all edges (and their weights) it has learned so far to all of its neighbors, receives (potentially new) edges, and repeats the introduction once more. As described from a high level in Section 2.6, we maintain the second invariant by employing a recursive divide-and-conquer approach. In the following, we describe how exactly the recursions are performed from the perspective of v.

We divide each phase into $\lceil\log n\rceil + 1$ steps, each of which corresponds to one level of recursion. Throughout the execution of these steps, v maintains a set R of *recursion messages*. Initially, R only contains the message $\langle v, \hat{d}(v), \emptyset\rangle$, which initiates v's own recursion on $T(v, i)$ (recall that $T(v, i)$ is the shortest-path tree of $G(v, i)$ rooted at v that contains only shortest paths with a minimum number of hops).

Each recursion message $\langle u, d_{uv}, L\rangle \in R$ corresponds to a recursion that v currently handles, and is associated with a node u within hop-distance i to v that originally initiated the recursion on $T(u, i)$ (u might be v itself). The reason for v now storing this message is because in some preceding step, u instructed v to continue the recursion on a subtree T of $T(u, i)$ that is rooted at v (we call T the corresponding *recursion subtree*). Further, the recursion message contains a value $d_{uv} := \hat{d}(u) + d_{T(u,i)}(u, v)$, which is the length of a path from s to v that contains u. Finally, L is a subset of nodes of $T(u, i)$ from which v can infer T. More precisely, T is the subtree of $T(u, i)$ rooted at v without all subtrees rooted at any node of L. Note that whereas we cannot efficiently send T over the global network, T can be inferred from v's knowledge of $T(u, i)$ (which is fully contained in $G(v, 2i)$), and the set L, which will contain at most $O(\log n)$ nodes.

In each step, all recursion messages are processed by v simultaneously. First, v updates $\hat{d}(v)$ to the minimum of $\hat{d}(v)$ and all distance values contained in R. Consider $\langle u, d_{uv}, L\rangle \in R$ at the beginning of some step. If $|T| \leq 1$ (i.e., T contains at most one node), then there is no need to continue the recursion. Otherwise, v selects a *splitting node* whose removal disconnects T into components of size $|T|/2$; these components become recursion subtrees in the next step. Let T_x be the subtree of T rooted at x, then the resulting components are (i) $T\setminus T_x$, which is rooted at v, and (ii) all subtrees of T rooted at a child of x in T. To continue the recursion in all components, v needs to send a recursion message to the root of each component.

For the component of (i), v simply sends a recursion message $(u, d_{uv}, L \cup \{x\})$ to itself; this message instructs v to continue the recursion on $T \setminus T_x$. For (ii), v needs to send a recursion message $\langle u, d_{uc}, \emptyset\rangle$, where $d_{uc} := d_{uv} + d_{T(u,i)}(v, c)$, to every child c of x in T_x. However, x may have up to $\Theta(n)$ children; therefore, instead of sending recursion messages directly, v instructs x to forward the respective messages to its children by sending a *splitting message* $\langle u, d_{ux}, sph_{T(u,i)}(u, x)\rangle$ to x (which contains node u, the length $d_{ux} := d_{uv} + d_{T(u,i)}(v, x)$ of a path from s to x that contains u and v, and the number of hops from u to x in $T(u, i)$). Since x can infer $T(u, i)$ due to Invariant (i), x can reconstruct all recursion messages it is supposed to forward to its children in T_x and send them directly using *local* edges.

As already pointed out in Section 2.6, v cannot send its splitting message to x directly, as x may be the recipient of many splitting messages in this step. However, it suffices for every node to only receive the splitting message that contains the smallest distance value among all splitting messages destined at it in this step. If there are several splitting messages with the same distance value, we prefer the one with smaller hop value, breaking ties by choosing the message that contains the node with smallest identifier. In Lemma 5.2, we will argue that for every node v with $sph(s, v) \leq t(i)$ there must be a node u such that $\hat{d}(u)+d_{T(u,i)}(u, v) = d(s, v)$ and every recursion message corresponding to u that is destined at a node of the branch from u to v in $T(u, i)$ is successfully delivered; therefore, v will learn $d(s, v)$ anyway.

Thus, the nodes send all splitting messages of this step using Theorem 2.2 of [4], which we can use as a black box in the global network since it is designed for the node-capacitated clique model. Specifically, all splitting messages destined at the same node v translate into an *aggregation group*, for which the algorithm combines all messages into only the message with smallest associated distance value. After $O(\log n)$ rounds, w.h.p., the message is delivered to its recipient. When v receives a splitting message $\langle u, d_{uv}, sph_{T(u,i)}(u, v)\rangle$ in this way, it updates $\hat{d}(v)$ to $\min\{\hat{d}(v), d\}$ and sends the corresponding recursion messages to its children in $T(u, i)$. If v is adjacent to multiple nodes that have received splitting messages, it may also receive multiple recursion messages; again, it suffices for v to only store the recursion message that contains the smallest distance value. We again break ties by preferring the recursion message associated with the node u that minimizes $sph_{T(u,i)}(u, v)$ (which u can infer from the knowledge of $T(u, i)$) or, in case of a tie, has smallest identifier.

Note that the set of recursion messages R' stored by v at the beginning of the next phase contains at most one recursion message for each message in R (i.e., if the corresponding recursion needs to be continued at v), and at most one additional recursion message that v received from one of its neighbors. However, as the size of the subtree of each recursion halves in each step, there are no recursion messages after $\lceil \log n\rceil + 1$ steps anymore, and v continues with the next phase.

As we do not require the nodes to know SPD, we have to let the nodes detect when to terminate. We simply stop the algorithm when for the first time no distance estimate changes at any node. This can be detected by simply performing a convergecast in the global network (see, e.g., [4]) at the end of every phase in time $O(\log n)$. When for the first time no value changed anymore, all nodes terminate.

We begin our analysis by showing that each recursion subtree halves in size every step, which implies that after $\lceil \log n \rceil + 1$ steps no node stores a recursion message anymore. The proof of the following lemma can be found in the full version [5].

Lemma 5.1. *Let T be a recursion subtree with $|T| \geq 2$. Node v can compute a splitting node x whose removal disconnects T into trees each of size at most $|T|/2$.*

The next three lemmas prove Invariant (ii), which implies that eventually every node v knows $d(s,v)$, the correctness of termination, and the runtime of the algorithm. The proofs can be found in the full version [5].

Lemma 5.2. *Let $v \in V$ such that $sph(s,v) \leq t(i-1)$. $\hat{d}(v) = d(s,v)$ at the beginning of phase i.*

Lemma 5.3. *No distance estimate changes in phase i if and only if $\hat{d}(v) = d(s,v)$ for every node $v \in V$.*

Lemma 5.4. *The algorithm terminates after $2\sqrt{\mathsf{SPD}}+1$ phases. Every phase takes time $O(\log^2 n)$, w.h.p.*

From the above, we conclude Theorem 2.6. The algorithm can easily be modified to solve (h,k)-SSP for given h and k. The following theorem is proven in the full version of the paper [5].

Theorem 5.1. *The modified algorithm solves (h,k)-SSP in time $\widetilde{O}(\sqrt{kh})$, w.h.p.*

Note that the above algorithm requires local capacity $\lambda = \Theta(n^2)$ as we might transfer the whole graph over a local edge in one round in the worst case.

5.2 Approximate SSSP in $\widetilde{O}(n^{1/3})$ Rounds. In the following, we give Algorithm 13 to compute a $(1+\varepsilon)$-approximation of SSSP in the claimed number of rounds.

Algorithm 13 `Approximate-SSSP(ε)`
$\triangleright$ *$h = \widetilde{O}(x)$, $1+\varepsilon$ is the approximation factor*

```
if $v$ is the source then mark $v$                ▷ ensure $s \in M$
else mark $v$ with probability $\frac{1}{x}$
for $h$ rounds do                 ▷ initially $v$ sends incident edges
    $v$ sends edges it learned last round to its neighbors
if $v$ is marked then
    participate in BCC simulation
    run algorithm of [7] on simulated BCC
    create token $\langle \mathrm{id}(v), \tilde{d}_{sv} \rangle$    ▷ $\tilde{d}_{sv}$ approx. of $d(s,v)$
Token-Dissemination           ▷ disseminate approx. dist.
approximate SSSP locally with Equation 5.4
```

$$\tilde{d}_{sv} := \min\Big(d_h(s,v), \min_{u \in M}\big(\tilde{d}_{su} + d_h(u,v)\big)\Big) \tag{5.4}$$

The base concept is to compute a skeleton graph $S = (M, E_S)$ (as in Section 4, whereas the source s is always included) and then use token dissemination to simulate the broadcast congested clique (BCC) model on M. In the BCC-model we have a synchronous message passing, where in every round each node can send one message of size $O(\log n)$, which is known by all nodes in the subsequent round. The simulation of the BCC-model on M allows the usage of the algorithm given in [7] for said model, to $(1+\varepsilon)$-approximate SSSP on S. Then we broadcast the distance estimations of all pairs $M \times \{s\}$ with token dissemination, which we can use to approximate SSSP-distances on the whole graph. A comprehensive discussion about the underlying concept and the properties of the algorithm, as well as the proof of Theorem 2.7 can be found in the full version [5].

5.3 Approximate SSSP in $\widetilde{O}(n^{\epsilon})$ Rounds. In this section we present a fast algorithm that runs in $O(\alpha^3)$ for some parameter $\alpha \geq 5$, albeit with a coarser approximation ratio of $(\log_\alpha n)^{O(\log_\alpha n)}$. Nevertheless, notice that we get a constant approximation ratio when we set $\alpha = n^\epsilon$ for fixed $\epsilon > 0$. The key ingredient for our algorithm is to compute a sparse spanner of the skeleton graph that we call a *skeleton spanner*, and which we formally define shortly. In the first part of this section, we present an algorithm to compute a skeleton spanner. Subsequently, we describe how this algorithm can be used to compute $(\log_\alpha n)^{O(\log_\alpha n)}$-approximate SSSP by constructing a hierarchy of skeleton spanners.

Constructing a Skeleton Spanner. We first define skeleton spanners fromally in Definition 1 and subsequently describe the algorithm to compute such a spanner from a high level. As the algorithm is solely based on computing *limited-depth Bellman-Ford computations*, we can efficiently execute it in the global network by using the methods of [4], which we describe afterwards.

Definition 1. (Skeleton Spanner) *Let $G = (V,E,w)$ be a weighted graph, let $M \subseteq V$ be a set of marked nodes of G, and let $h \in \mathbb{Z}_{\geq 1}$. An h-*hop skeleton spanner *$H = (M, E_H)$ with stretch $s \geq 1$ is a weighted graph with the following properties:*

1. *Every edge $\{u,v\} \in E_H$ corresponds to a path P in G between u and v such that $w(u,v) = w(P)$.*
2. *For every two nodes $u,v \in M$, we have $d_{h,G}(u,v) \leq d_H(u,v) \leq s \cdot d_{h,G}(u,v)$.*

We now present our algorithm to construct a skeleton spanner from a high level. Assume that we are given a graph $G = (V,E,w)$, a set of marked nodes $M \subseteq V$,

Algorithm 14 `Skeleton-Spanner` ▷ $\eta > 1$, $k \geq 2$
▷ *Stage i dealing with h-lim. distances $\in [L_i/\eta, L_i]$*

```
$V_0 := V$    ▷ $V_j$ is set of nodes active in phase $j$
for $j := 0$ to $k-1$ do
    $G_j := G[V_j]$    ▷ subgraph of $G$ induced by $V_j$
    for each $r \in V_j \cap M$ do
        randomly sample $r$ with probability $|M|^{\frac{j+1}{k}-1}$
    for each sampled $r \in M$ do
        for each $v \in B_{G_j}(r, k-j, L_i) \cap M$ do
            add $\{v, r\}$ of weight $d_{h(k-j),G_j}(v, r)$ to $E_H$
    $V_{j+1} := V_j \setminus B_{G_j}(r, k-j-1, L_i)$    ▷ $r$ sampled
```

and a hop-distance parameter $h \geq 1$. Let us further assume that for all $e \in E$, we have $1 \leq w(e) \leq W/h$ for some given $W \geq h$, so that the length of any path consisting of at most h hops is between 1 and W. The algorithm further has two parameters $k \geq 2$ and $\eta > 1$ that control the stretch and the number of edges of the resulting spanner.

The algorithm consists of $\lceil \log_\eta W \rceil$ stages. In the following, we focus on a specific stage $i \geq 1$. For convenience, we define $L_i := \eta^i$. The objective of stage i is to construct a subset of the edges of $H = (M, E_H)$ that provides a good approximation for any two nodes $u, v \in M$ for which the h-limited distance in G is in the range $[L_i/\eta, L_i]$. The final spanner is then obtained by taking the union of the edges computed in the individual stages.

Each stage consists of k phases, which we enumerate by $j = 0, 1, \ldots, k-1$. Initially, all nodes in M are active. We will show that nodes in M become inactive as soon as it is guaranteed that all their h-limited distances in the target range are approximated well enough. In the following, for a node $r \in G$, an integer parameter $x \geq 1$, and a distance $L \geq 1$, we define the *ball* of r as

$$B_G(r, x, L) := \{v \in V \,:\, d_{h \cdot x, G}(r, v) \leq x \cdot L\}.$$

The details of the algorithm for stage i are given in Algorithm 14. We refer to the k iterations of the outermost for-loop as the k phases $j = 0, \ldots, k-1$.

The following sequence of lemmas proves that the algorithm constructs an h-hop skeleton spanner. All missing proofs of this section can be found in the full version [5].

Lemma 5.5. *When a node $u \in M$ gets deactivated in stage i, for every $v \in M$ for which $d_{h,G}(u, v) \leq L_i$, the algorithm has added a path of length at most $2kL_i$ to the spanner edge set E_H. Furthermore, this path consists of at most 2 edges.*

The next lemma shows that in each phase, every node is only within the ball of few random centers. On the one hand, this implies that the spanner algorithm does not add too many edges; on the other hand, it also allows to execute the algorithm efficiently by using only global edges. The lemma follows because the radius of the balls decreases from phase to phase, and the radius in which nodes are deactivated in phase j is the same as the radius of the ball of nodes to which edges are established in phase $j+1$. Therefore, roughly speaking, if a node v expects to "see" many centers in phase $j+1$ (to which it would establish an edge), the node should have been deactivated in phase j. A similar argument has previously been used by Blelloch et al. in [9].

Lemma 5.6. *In every phase j of Algorithm 14, every node $v \in V_j$ is in $B_{G_j}(r, k-j, L_i)$ for at most $O(|M|^{1/k} \log n)$ sampled nodes $r \in M$, w.h.p.*

We now have everything we need in order to prove the main property of the described spanner algorithm. More specifically, the lemma can be shown by combining the following observations: (1) for any two nodes $u, v \in M$ with $d_{h,G}(u, v) \leq L_i$, a (≤ 2)-hop path of length $2\eta k \cdot d_{h,G}(u, v)$ is created in stage i by Lemma 5.5, and (2) in each phase of each stage, every node in M adds at most $O(|M|^{1/k} \log n)$ edges to E_H by Lemma 5.6.

Lemma 5.7. *Given a weighted graph $G = (V, E, w)$, a set of marked nodes $M \subseteq V$, as well as parameters $h \geq 1$, $k \geq 2$, and $\eta > 1$, the described spanner algorithm computes an h-hop skeleton spanner $H = (M, E_H)$ with stretch $2\eta k$. $|E_H| = O(k \cdot |M|^{1+1/k} \log n \cdot \log_\eta W)$, w.h.p. Further, for any two nodes $u, v \in M$ such that $hop_G(u, v) \leq h$ we have that $d_{2,H}(u, v) \leq 2\eta k \cdot d_{h,G}(u, v)$ (i.e., there exists a path P with at most 2 hops and length at most $2\eta k \cdot d_{h,G}(u, v)$).*

It remains to show how the algorithm can be efficiently implemented in our model. As we will perform the algorithm only in the global network, we can again use techniques from [4]. We assume the graph G on which we aim to construct an h-hop skeleton spanner H is given in *δ-oriented form.* We say a graph is in δ-oriented form if every edge $\{u, v\}$ is only known by one of its endpoints (we say that endpoint is *responsible* for the edge), such that every node in G is responsible for at most δ edges. We also construct H in such a form; more specifically, when $v \in B_{G_j}(r, k-j, L_i) \cap M$ in phase j and stage i of the algorithm, the edge $\{v, r\}$ is added to E_H by v and without the knowledge of r, and v becomes responsible for the edge. We first describe the communication primitive we employ to implement the algorithm efficiently, and then explain how precisely the primitive is used.

To efficiently execute Algorithm 14 in the global network, we follow the idea of [4]: instead of letting neighbors in G_j (with potentially high degree) send messages to each other *directly*, we use a random load balancing technique that exploits the fact that every node is responsible for at most δ edges. Specifically, in our algorithm the nodes exclusively communicate by performing *multi-aggregations* (see [4], Theorem 2.5). In a multi-aggregation, every node v that wants to communicate a message to its neighbors in G_j injects the message into a *multicast tree* $M_{v,j}$, which is a subtree of a butterfly network that, roughly speaking, connects v with all of its neighbors in G_j. In $M_{v,j}$, one copy of the message is created for each of v's neighbors in G_j. All messages that are destined at the same node u in G_j are combined using an aggregate function, so that u eventually receives a single message containing this aggregate. If at the beginning of phase j all $M_{v,j}$ are setup for all $v \in V_j$ in an appropriate way, then Theorem 2.5 of [4] implies that the multi-aggregation can be carried out in time $O(C + \log n)$, w.h.p. Here, C is the so-called *congestion* of the trees, which, as we formally show in the following lemma, is only $O(\delta + \log n)$ in our case.

LEMMA 5.8. *There is an algorithm that constructs a multicast tree $M_{v,j}$ for every source $v \in V_j$, whose multicast group is the set of its neighbors in G_j, with congestion $O(\delta + \log n)$ and in time $O(\delta + \log n)$ at the beginning of phase j, w.h.p. Furthermore, the leaf of $M_{v,j}$ that corresponds to v's neighbor u learns $w(u,v)$.*

We now describe how multi-aggregations can be used to execute Algorithm 14. Specifically, we need to ensure that in phase j every node $v \in V_j$ learns $d_{h(k-j),G_j}(v,r)$ for each sampled $r \in M$ if $d_{h(k-j),G_j}(v,r) \le (k-j)L_i$. From a high level, we perform a distributed Bellman-Ford algorithm in G_j from every source $r \in M$ that is sampled in phase j. More precisely, in iteration t of the algorithm, every node $v \in V_j$ learns $d_{t,G_j}(r,v)$ for every node $r \in R_{v,t} := \{u \in M \mid u \text{ is sampled and } d_{t,G_j}(r,v) \le (k-j)L_i\}$. Therefore, after $h(k-j)$ iterations, v knows whether it is in $B_{G_j}(r, k-j, L_i)$, in which case it adds $\{v,r\}$ to E_H and becomes responsible for it. Further, if $v \in B_{G_j}(r, k-j-1, L_i)$, then $d_{h(k-j-1),G_j}(r,v) \le (k-j-1)L_i \le (k-j)L_i$, and thus v can also infer whether it becomes inactive in this phase.

In iteration t of the algorithm, every node v performs a (slightly modified) multi-aggregation to inform each of its neighbors about $d_{t-1,G_j}(r,v)$ for every node $r \in R_{v,t}$. Using the fact that the leaf of $M_{v,j}$ that corresponds to v's neighbor u knows $w(u,v)$ for all u by Lemma 5.8, we can modify the multi-aggregation so that v only receives

$$\min_{u \in V_j} \left(d_{t-1,G_j}(r,u) + w(u,v)\right) = d_{t,G_j}(r,v)$$

for each $r \in R_{v,t}$. Here, distances that grow too large will simply be dropped by the respective leaf node, and the multi-aggregation needs to be slowed down by a factor linear in the the maximum size of $R_{v,t}$ for all v. However, since $R_{v,t}$ is a subset of all nodes for which v is in $B_{G_j}(r, k-j, L_i)$, $|R_{v,t}| = O(|M|^{1/k} \log n)$ by Lemma 5.6. The details of our modified multi-aggregation can be found in the proof of the following lemma.

LEMMA 5.9. *Suppose $G = (V, E, w)$ is a weighted subgraph of the global network given in δ-oriented form and $M \subseteq V$. Then, an h-hop spanner as described in Lemma 5.7 in $O(k \cdot |M|^{1/k} \log n \cdot \log_\eta W)$-oriented form can be constructed in time $O((\delta + \log n)|M|^{1/k} \log n \cdot hk^2 \cdot \log_\eta W)$, w.h.p.*

The Recursive Algorithm. Using the sparse skeleton spanner algorithm, we now present the algorithm to approximate SSSP. The algorithm is divided into two stages. The purpose of the first stage is to compute a hierarchical structure of spanners $G_1, \dots, G_T$ as follows. Let $G_0 := G$, and choose parameters $\alpha \ge 5$, $h := c\alpha$ for a sufficiently large constant c, $k := \log_\alpha n$, and constant $\eta > 1$. We construct the first sparse spanner G_1, which contains all nodes of G, by performing the distributed Baswana-Sen algorithm [6] in the local network with parameter k, where any time a node adds an edge to the spanner, it becomes responsible for that edge. It can be shown that thereby we obtain a $(2\log_\alpha n - 1)$-spanner in $\widetilde{O}(\alpha)$-oriented form, w.h.p., in time $O(\log_\alpha^2 n)$. Every other spanner G_i ($i \ge 2$) is constructed as an h-hop skeleton spanner of G_{i-1}, where every node in G_{i-1} joins the set M_i of marked nodes with probability $\log(n)/\alpha$ for $i = 2$ and with probability $1/\alpha$ for $i \ge 3$. When for the first time a spanner G_{T+1} contains no nodes anymore, the first stage of the algorithm ends.

After the first stage has finished, in the second stage we simply perform a distributed Bellman-Ford algorithm for source s in the union of all recursively constructed spanners $H = \bigcup_{1 \le j \le T} G_i$ for $O(\alpha \log_\alpha n)$ rounds. Finally, every node $v \in V$ chooses the minimum of all received distance values as its estimate $\widetilde{d}(s,v)$ of $d(s,v)$. In the following, we show that H is a good spanner of the underlying graph G and that, moreover, between any two nodes of G, there is a path consisting of at most $O(\alpha \log n)$ hops in H, whose length gives a good distance approximation of the actual length of a shortest path in G. We first need a technical lemma.

LEMMA 5.10. *Assume that P is a shortest path on G_1 between two nodes of G_1. Further consider $i \geq 2$ and let $u, v \in M_i$ be two nodes on the path P that are within q hops for some $q \in [\gamma \cdot \alpha^{i-2}, \gamma \cdot \alpha^{i-1}]$ for a sufficiently large constant γ. Then, u and v are connected in G_{i-1} by a path P such that P consists of at most $O(\alpha)$ hops and that has length at most $(2\eta k)^{i-1} \cdot d_{G_1}(u, v)$.*

We next prove that in the union spanner graph H, there is a short (in terms of hops and length) path between any two nodes.

LEMMA 5.11. *Let $u, v \in V$ be two nodes of G_1 and let P be a shortest path between u and v on G_1. Assume that P consists of q hops and assume that for the constant γ from Lemma 5.10, ξ is the smallest integer for which $q \leq \gamma\alpha^{\xi}$. Then graph H contains a path that consists of at most $O(\alpha \log_\alpha n)$ hops and that has length at most $(2\eta k)^{\xi-1} d_{G_1}(u, v)$.*

We are now ready to conclude Theorem 2.8. The runtime follows from summing up the time needed to construct all skeleton spanners, and the fact that Bellman-Ford needs only $O(\alpha \log_\alpha n)$ iterations, where each iteration can be performed efficiently in H using techniques from [4] and exploiting the fact that H is in $\widetilde{O}(\alpha)$-oriented form.

References

[1] A. Abboud, K. Censor-Hillel, and S. Khoury. Near-linear lower bounds for distributed distance computations, even in sparse networks. In *30th International Symposium on Distributed Computing (DISC)*, pages 29–42, 2016.

[2] U. Agarwal, V. Ramachandran, V. King, and M. Pontecorvi. A deterministic distributed algorithm for exact weighted all-pairs shortest paths in $\widetilde{O}(n^{3/2})$ rounds. In *Proc. of the 2018 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 199–205, 2018.

[3] J. N. Al-Karaki, G. A. Al-Mashaqbeh, and S. M. Bataineh. Routing protocols in wireless mesh networks: a survey. *Int. J. of Information and Communication Technology (IJICT)*, 11(4):445–495, 2017.

[4] J. Augustine, M. Ghaffari, R. Gmyr, K. Hinnenthal, F. Kuhn, J. Li, and C. Scheideler. Distributed computation in node-capacitated networks. In *Proc. of the 31st ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2019.

[5] J. Augustine, K. Hinnenthal, F. Kuhn, C. Scheideler, and P. Schneider. Shortest paths in a hybrid network model. *arXiv preprint arXiv:1909.01597*, 2019.

[6] S. Baswana and S. Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Structures & Algorithms*, 30(4):532–563, 2007.

[7] R. Becker, A. Karrenbauer, S. Krinninger, and C. Lenzen. Near-optimal approximate shortest paths and transshipment in distributed and streaming models. In *31st International Symposium on Distributed Computing (DISC)*, pages 7:1–7:16, 2017.

[8] A. Bernstein and D. Nanongkai. Distributed exact weighted all-pairs shortest paths in near-linear time. In *51st ACM Symposium on the Theory of Computing (STOC)*, 2019.

[9] G. E. Blelloch, A. Gupta, I. Koutis, G. L. Miller, R. Peng, and K. Tangwongsan. Nearly-linear work parallel SDD solvers, low-diameter decomposition, and low-stretch subgraphs. *Theory of Comput. Syst.*, 55(3):521–554, 2014.

[10] K. Censor-Hillel, M. Dory, J. H. Korhonen, and D. Leitersdorf. Fast approximate shortest paths in the congested clique. *arXiv preprint arXiv:1903.05956*, 2019.

[11] K. Censor-Hillel, P. Kaski, J. H. Korhonen, C. Lenzen, A. Paz, and J. Suomela. Algebraic Methods in the Congested Clique. In *Proc. of the 2015 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 143–152. ACM, 2015.

[12] A. Drucker, F. Kuhn, and R. Oshman. On the power of the congested clique model. In *Proc. of the 33rd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 367–376, 2014.

[13] M. Elkin. Distributed exact shortest paths in sublinear time. In *49th Annual ACM Symposium on Theory of Computing (STOC)*, pages 757–770, 2017.

[14] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat. Helios: a hybrid electrical/optical switch architecture for modular data centers. In *Proc. of the ACM SIGCOMM 2010 Conference*, pages 339–350, 2010.

[15] K. Foerster, M. Ghobadi, and S. Schmid. Characterizing the algorithmic complexity of reconfigurable data center architectures. In *2018 Symposium on Architectures for Networking and Communications Systems (ANCS)*, pages 89–96, 2018.

[16] S. Forster and D. Nanongkai. A faster distributed single-source shortest paths algorithm. In *59th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 686–697, 2018.

[17] S. Frischknecht, S. Holzer, and R. Wattenhofer. Networks cannot compute their diameter in sublinear time. In *23rd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1150–1162, 2012.

[18] M. Ghaffari and J. Li. Improved distributed algorithms for exact shortest paths. In *50th ACM Symposium on Theory of Computing (STOC)*, pages 431–444, 2018.

[19] R. Gmyr, K. Hinnenthal, C. Scheideler, and C. Sohler. Distributed monitoring of network properties: The power of hybrid networks. In *Proc. of the 44th International Colloqium on Algorithms, Languages, and Programming (ICALP)*, pages 137:1–137:15, 2017.

[20] D. Halperin, S. Kandula, J. Padhye, P. Bahl, and

D. Wetherall. Augmenting data center networks with multi-gigabit wireless links. In *Proc. of the ACM SIGCOMM 2011 Conference*, pages 38–49, 2011.

[21] M. Henzinger, S. Krinninger, and D. Nanongkai. A deterministic almost-tight distributed algorithm for approximating single-source shortest paths. In *48th ACM Symposium on Theory of Computing (STOC)*, pages 489–498, 2016.

[22] S. Holzer and R. Wattenhofer. Optimal distributed all pairs shortest paths and applications. In *2012 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 355–364, 2012.

[23] C.-C. Huang, D. Nanongkai, and T. Saranurak. Distributed exact weighted all-pairs shortest paths in $\widetilde{O}(n^{5/4})$ rounds. In *58th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 168–179, 2017.

[24] D. Jung, C. Kolb, C. Scheideler, and J. Sundermeier. Competitive routing in hybrid communication networks. In *14th International Symposium on Algorithms and Experiments for Wireless Networks (ALGOSENSORS)*, 2018.

[25] T. Jurdzinski and K. Nowicki. MST in $O(1)$ rounds of congested clique. In *Proc. of the 29th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2018.

[26] U. N. Kar and D. K. Sanyal. An overview of device-to-device communication in cellular networks. *ICT Express*, 4(3):203–208, 2018.

[27] C. Lenzen. Optimal deterministic routing and sorting on the congested clique. In *Proc. of the 32nd Symposium on Principles of Distributed Computing (PODC)*, pages 42–50, 2013.

[28] C. Lenzen and B. Patt-Shamir. Fast routing table construction using small messages. In *45th ACM Symposium on Theory of Computing (STOC)*, pages 381–390, 2013.

[29] C. Lenzen and B. Patt-Shamir. Fast partial distance estimation and applications. In *2015 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 153–162, 2015.

[30] C. Lenzen and D. Peleg. Efficient distributed source detection with limited bandwidth. In *2013 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 375–382, 2013.

[31] Z. Lotker, B. Patt-Shamir, E. Pavlov, and D. Peleg. Minimum-weight spanning tree construction in $O(\log\log n)$ communication rounds. *SIAM J. on Computing*, 35(1):120–131, 2005.

[32] D. Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *46th ACM Symposium on Symposium on Theory of Computing (STOC)*, pages 565–573, 2014.

[33] D. Peleg, L. Roditty, and E. Tal. Distributed algorithms for network diameter and girth. In *Proc. of the 39th International on Colloquium Automata, Languages, and Programming (ICALP)*, pages 660–672, 2012.

[34] M. Rossberg and G. Schaefer. A survey on automatic configuration of virtual private networks. *Computer Networks*, 55(8):1684–1699, 2011.

[35] A. D. Sarma, S. Holzer, L. Kor, A. Korman, D. Nanongkai, G. Pandurangan, D. Peleg, and R. Wattenhofer. Distributed verification and hardness of distributed approximation. *SIAM J. on Computing*, 41(5):1235–1265, 2012.

[36] A. Tell, W. Babalola, G. Kalebiala, and K. Chinta. Sd-wan: A modern hybrid-wan to enable digital transformation for businesses. *IDC White Paper*, April 2018.

[37] J. D. Ullman and M. Yannakakis. High-probability parallel transitive-closure algorithms. *SIAM J. on Computing*, 20(1):100–125, 1991.

A General Notions from Probability Theory

LEMMA A.1. (CHERNOFF BOUND) *We use the following forms of Chernoff bounds in our proofs:*

$$\mathbb{P}\big(X > (1+\delta)\mu_H\big) \le \exp\big(-\frac{\delta\mu_H}{3}\big),$$

with $X = \sum_{i=1}^n X_i$ for i.i.d. random variables $X_i \in \{0,1\}$ and $\mathbb{E}(X) \le \mu_H$ and $\delta \ge 1$. Similarly, for $\mathbb{E}(X) \ge \mu_L$ and $0 \le \delta \le 1$ we have

$$\mathbb{P}\big(X < (1-\delta)\mu_L\big) \le \exp\big(-\frac{\delta^2\mu_L}{2}\big).$$

LEMMA A.2. (UNION BOUND) *Let $E_1, \dots, E_k$ be events, each taking place w.h.p. If $k \le p(n)$ for a polynomial p then $E := \bigcap_{i=1}^k E_i$ also takes place w.h.p.*

Proof. Let $d := \deg(p)+1$. Then there is an $n_0 \ge 0$ such that $p(n) \le n^d$ for all $n \ge n_0$. Let $n_1, \dots, n_k \in \mathbb{N}$ such that for all $i \in \{1, \dots, k\}$ we have $\mathbb{P}(\overline{E_i}) \le \frac{1}{n^c}$ for some (yet unspecified) $c > 0$. With Boole's Inequality (union bound) we obtain

$$\mathbb{P}\big(\overline{E}\big) = \mathbb{P}\Big(\bigcup_{i=1}^k \overline{E_i}\Big) \le \sum_{i=1}^k \mathbb{P}(\overline{E_i}) \le \sum_{i=1}^k \frac{1}{n^c} \le \frac{p(n)}{n^c} \le \frac{1}{n^{c-d}}$$

for all $n \ge n_0' := \max(n_0, \dots, n_k)$. Let $c' > 0$ be arbitrary. We choose $c \ge c' + d$. Then we have $\mathbb{P}\big(\overline{E}\big) \le \frac{1}{n^{c'}}$ for all $n \ge n_0'$. ◻

REMARK 1. *If a finite number of events is involved we use the above lemma without explicitly mentioning it. It is possible to use the lemma in a nested fashion as long as the number of applications is polynomial in n.*