On the complexity of distributed BFS in ad hoc networks with spontaneous wake-up.

Dariusz R. Kowalski, Krzysztof Krzywdziński

Discrete Mathematics and Theoretical Computer Science DMTCS vol. **15:3**, 2013, 101–118

On the complexity of distributed BFS in ad hoc networks with non-spontaneous wake-ups

Dariusz R. Kowalski[1†] Krzysztof Krzywdziński[2‡]

[1]*Department of Computer Science, University of Liverpool, Liverpool, United Kingdom*
[2]*Faculty of Mathematics and Computer Science, Adam Mickiewicz University, Poznan, Poland*

received 21st Oct. 2012, revised 31st Aug. 2013, accepted 31st Oct. 2013.

We study time and message complexity of the problem of building a BFS tree by a spontaneously awaken node in ad hoc network. Computation is in synchronous rounds, and messages are sent via point-to-point bi-directional links. Network topology is modeled by a graph. Each node knows only its own id and the id's of its neighbors in the network and no pre-processing is allowed; therefore the solutions to the problem of spanning a BFS tree in this setting must be distributed. We deliver a deterministic distributed solution that trades time for messages, mainly, with time complexity $O(D \cdot \min(D, n/f(n)) \cdot \log D \cdot \log n)$ and with the number of point-to-point messages sent $O(n \cdot (\min(D, n/f(n)) + f(n)) \cdot \log D \cdot \log n)$, for any n-node network with diameter D and for any monotonically non-decreasing sub-linear integer function f. Function f in the above formulas come from the threshold value on node degrees used by our algorithms, in the sense that nodes with degree at most $f(n)$ are treated differently that the other nodes. This yields the first BFS-finding deterministic distributed algorithm in ad hoc networks working in time $o(n)$ and with $o(n^2)$ message complexity, for some suitable functions $f(n) = o(n/\log^2 n)$, provided $D = o(n/\log^4 n)$.

Keywords: BFS tree, ad hoc network, distributed algorithm, message complexity

1 Introduction

We consider a message-passing distributed system consisting of n processes, also called nodes, with pairwise different id's. There is an underlying network of point-to-point connections between processes, modeled by an undirected graph $G = (V, E)$. Each node knows only its own id and the id's of its neighbors in the network; this knowledge is however required only for the purpose of distinguishing local ports through which messages are sent. We do not make any specific assumption about the domain of ids, apart that ids are pairwise disjoint and small enough to be contained in messages and to be processed locally. We denote by D the diameter of the underlying network graph G. Computation and communication is in synchronous rounds, each consisting of three parts: receiving messages that have been sent to the

†Email: D.Kowalski@liverpool.ac.uk.
‡Email: kkrzywd@amu.edu.pl. Research supported by NCN grant N N206 565740. Research partialy supported by grant N206 017 32/2452.

node in the previous round (if any), local computation, and sending messages via selected point-to-point links. Each message may carry the original information and a polynomial number of additional bits (our algorithms use in fact $O(n^2 \log n)$ additional bits per message). The network is reliable, in the sense that there are no failures or delays in local computations and point-to-point message propagation.

In this work we consider the task of finding a BFS tree (i.e., a spanning tree consisting of some shortest paths from the root to other nodes) in the underlying network. The task occurs in an arbitrary node, while all other nodes are not aware of it. During the computation the network topology does not change. The resulting tree should be rooted at the node in which this task has occurred. Observe that due to ad hoc setting with non-spontaneous wake-ups, i.e., all nodes except the initially awaken one could join the computation only after receiving a message from some node already participating in the execution assumed in this work, a solution to this problem must be distributed. We focus only on deterministic solutions. We consider a single task of building a BFS tree, which occurs in an arbitrary single node.[(i)]

Our results and the structure of this work. We deliver a deterministic distributed BFS-finding algorithm that trades time for messages. More precisely, our algorithm solves the problem with time complexity $O(D \cdot \min(D, n/f(n)) \cdot \log D \cdot \log n)$ and with the number of point-to-point messages being $O(n \cdot (\min(D, n/f(n)) + f(n)) \cdot \log D \cdot \log n)$, for any monotonically non-decreasing sub-linear integer function f. This yields the first deterministic distributed BFS-finding algorithm in ad hoc networks working in time $o(n)$ and with $o(n^2)$ message complexity, for some suitable function $f(n) = o(n/\log^2 n)$, provided $D = o(n/\log^4 n)$. In our algorithm, function f defines the threshold value for node degree, e.g., nodes with degree at most $f(n)$ are treated differently than the other nodes.

Section 2 presents a flooding-like deterministic distributed procedure Shallow_Tide, which propagates information along paths of bounded length and bounded node degree, and then collects the information backwards. We use this routine for designing three deterministic distributed algorithms computing a BFS tree in the underlying network G, differing on input specification, c.f., Sections 3 and 4.

Previous and related work. A naive distributed implementation of an algorithm computing a BFS tree in ad hoc network is by flooding a message along all edges of the network and choosing a parent arbitrarily from the senders of the received messages. This approach however requires D rounds and $|E|$ point-to-point messages, which in case of dense graphs gives $\Theta(n^2)$ message complexity (folklore result). BFS tree can also be built layer by layer, each time by forwarding information about neighborhoods of the front layer nodes to the root, which then computes BFS edges to connect the nodes in the next layer and forwards this information to the nodes in the front layer. This method requires time $\Theta(D^2)$ and $\Theta(nD)$ messages to be sent, which for dense or shallow graphs can be smaller that $|E|$ messages sent by the flooding algorithm.

Another method of spanning a BFS tree is to apply a gathering type of algorithms that collect the information about the underlying network topology in one node and compute a BFS tree locally (see e.g., the monographs Hromkovic et al. (2005); Peleg (2000)). For example, a distributed implementation of a DFS search algorithm gathering such information about the underlying network topology requires $\Theta(n)$ rounds and $\Theta(n)$ messages to be sent. This result, comparing with the previously described upper bounds, raises the following question, addressed in this work: how the number of messages sent in the process of spanning a BFS tree depends on time complexity.

[(i)] In general setting of many spontaneously arriving tasks, they can be run in parallel at nodes, and the cost of each such task can be bounded as in this work.

A BFS tree and other related problems were widely studied in an asynchronous setting, c.f., Awerbuch (1987); Awerbuch and Gallager (1985, 1987); Awerbuch et al. (1990). In this model, however, there is no scope for tradeoff between time and message complexity, because there is no notion of rounds. More precisely, a lower bound $\Omega(|E|)$ on message complexity was proved in Awerbuch et al. (1990), and a super-linear, in terms of n, lower bound $\Omega(n \log n)$ was shown in Korach et al. (1989). The problem of efficient maintaining of a BFS tree in dynamic networks was considered in Awerbuch et al. (2008), where the amortized cost of $O(n)$ messages per single change in the topology was obtained. However, we are not aware of any deterministic distributed or randomized solution for the BFS-finding problem (or the related gathering problem) in the model of ad hoc networks with non-spontaneous wake-ups that would achieve both: sublinear time complexity $o(n)$ and subquadratic message complexity $o(n^2)$, for any network of diameter $o(n/\log^4 n)$.

Another way of propagating/gathering information is by epidemic broadcast/convergecast Demers et al. (1988). In this method, a node selects a small, typically random, sample among its neighbors to push/pull the information; see also Kempe et al. (2004); Boyd et al. (2006) for other randomized communication algorithms. The problem of finding a BFS tree was considered also in other network settings different from the classic message-passing model. For example, in *radio networks*, where collisions among messages arriving at the same node at the same round are possible, the problem of deterministic broadcasting (which is no more difficult than building a BFS tree) requires overlinear, in the number of nodes, time, c.f., Chrobak et al. (2006); Clementi et al. (2001). Fault-tolerant communication in networks was also studied in various models and scenarios, see for example Chlebus and Kowalski (2006) for results and references regarding all-to-all communication in fully-connected message passing system with dynamic crashes, Bienkowski et al. (2010) for recovering from crashes in networks of general topology, and Hedetniemi et al. (1988); Hromkovic et al. (2005); Pelc (1996) for a number of fault-tolerant communication algorithms in general networks. In this context, our work can be viewed as recovering a communication structure in network after accidental failures. Our algorithms are the first that span a tree in ad hoc setting, without any pre-processing, in time $o(n)$ and using $o(n^2)$ point-to-point messages, for $D = o(n/\log^2 n)$.

2 Procedure Shallow_Tide

One of the common methods of propagating information, especially in case of deterministic protocols, is by *flooding*, in which each node simply combines and re-sends a newly received information. This method however is not efficient from the perspective of the number of generated point-to-point messages, as in the worst-case scenario all point-to-point links may be used for propagating a single broadcast request. Our flooding-type procedure, called Shallow_Tide, first propagates information by building a partial BFS tree on some bounded and shallow part of the network, and then gathers the information backwards toward the source node, all using relatively small number of messages. More precisely, procedure Shallow_Tide$(s, \ell, \psi, \mathcal{T})$ triggered by a source node s, aims to

- wake-up nodes of distance at most ℓ from the source, unless such a node is separated from the source by nodes of degree bigger than ψ that are not initially "known" by the source (i.e., by nodes of degree larger than ψ that are not in the input instance of $\mathcal{T}$);
- build, in a distributed way, a BFS tree rooted at source s on this part of the network;
- gather, in the source node s, the information about this tree and all the network links adjacent to nodes in the tree.

SHALLOW_TIDE$(s, \ell, \psi, \mathcal{T})$ — procedure for node $v = s$:

1. Round 0:

   Receive: no message

   Compute: initialize $\mathcal{H} \leftarrow (\{s\} \cup N(s), \{\{s, v\} : v \in N(s)\})$ and *counter* $\leftarrow 0$

   Send: message $(\textit{counter} + 1, s, \ell, \psi, \mathcal{T}, \mathcal{H})$ to all nodes in $N(s)$

2. Stay idle for 2ℓ rounds

3. Backward collection round:

   Receive: all point-to-point messages $(\textit{counter}', s, \ell, \psi, \mathcal{T}', \mathcal{H}')$ sent to node s in sending part of the preceding round

   Compute: update local copy of $\mathcal{T}$ by adding new nodes and edges from the received copies of $\mathcal{T}'$;
   update local copy of $\mathcal{H}$ by adding new nodes and edges from the received copies of $\mathcal{H}'$;

   Send: no message

OUTPUT: $\mathcal{T}, \mathcal{H}$

Fig. 1: Procedure Shallow_Tide$(s, \ell, \psi, \mathcal{T})$: pseudo-code for node s.

The procedure is of "wake-up" type, i.e., the source itself triggers the procedure while all other nodes stay dormant until they get a message generated by this procedure run at one of the neighboring nodes. We call all nodes that receive such a message *participating nodes*. More details follow.

The input of procedure Shallow_Tide contains the following parameters:

- the source node s initializing the procedure;
- upper bound ℓ on the maximal depth of flooding;
- threshold value ψ to be compared with the degree of a node in the network; if the degree is bigger than ψ then the received and updated message is not re-sent by that node;
- tree $\mathcal{T}$, rooted at node s, being a subgraph of the underlying network.

Additionally, a node v is equipped with set $N(v)$ of neighbors in the underlying network, provided to it as a part of the model setting.

Messages sent in this protocol carry the following information:

- round counter: it helps to set the local counter of receiving nodes, according to the round number after starting the procedure at the source node s; initially set by source s to 0;
- all input parameters s, ℓ, ψ: in order to be able to initialize the procedure in new nodes;

- variable $\mathcal{T}$: an estimate (a subtree) of the BFS tree rooted at s; in the beginning of the protocol, node s initializes it to the corresponding tree $\mathcal{T}$ from the input of the procedure;
- variable $\mathcal{H}$: a graph being the current local estimate of the underlying network, initially set by source s into $(\{s\} \cup N(s), \{\{s, v\} : v \in N(s)\})$.

The idea of the procedure is as follows. The source sends a message to all its neighbors in the network in round 0, then it is idle for the next 2ℓ rounds, and in round $2\ell + 1$ it processes the received messages and updates its variables $\mathcal{T}, \mathcal{H}$. In the first ℓ rounds, a process $v \neq s$ that receives a message $(\mathit{counter}', s, \ell, \psi, \mathcal{T}', \mathcal{H}')$ for the first time updates its knowledge about $\mathcal{T}, \mathcal{H}$, sets its local counter, and re-sends a message either to its children in $\mathcal{T}$, if it is an internal node in $\mathcal{T}$, or to all its neighbors which were not included in the received graphs $\mathcal{H}'$ otherwise. This round is called a *joining round*. Then the node stays idle till the beginning of $(2(\ell - \mathit{counter}) + 1)$st round, when it receives messages from its children and re-sends a single message combining the received variables to its selected parent. See Figure 1 for the pseudo-code of procedure Shallow_Tide at node s and Figure 2 for the pseudo-code of procedure Shallow_Tide at node $v \neq s$.

2.1 Analysis of procedure Shallow_Tide

First note that, by simple inductive argument and the update rule for *counter* (c.f., steps 1(a), 1(f) and 1(g) in Figure 2), all nodes $v \neq s$ that receive their first message in round i, for $1 \leq i \leq \ell$, set their variables *counter* to i. Second, observe that all participating nodes, including the source, terminate in round at most $2\ell + 1$. Recall that we call a node participating in Shallow_Tide$(s, \ell, \psi, \mathcal{T})$ if it receives a message $(\mathit{counter}', s, \ell, \psi, \mathcal{T}', \mathcal{H}')$ in some round.

Denote the initial tree $\mathcal{T}$ by $\mathcal{T}_{in}$ and the outputted one by $\mathcal{T}_{out}$. Similarly, denote the outputted value of variable $\mathcal{H}$ by $\mathcal{H}_{out}$. For any given graph $\mathcal{H}$, let $V(\mathcal{H})$ denote the set of nodes of this graph. Let S be the set of nodes of degree at most ψ for which there exists a path of length at most ℓ from source s containing only nodes in $S \cup V(\mathcal{T}_{in})$. More precisely, define recursively: $S_0 = \{s\}$; for any $i \geq 1$, let S_{i+1} be the set of nodes v of degree at most ψ in graph G such that there is a path from s to v of length at most ℓ in the subgraph of G induced by the set of nodes $S_i \cup V(\mathcal{T}_{in})$; let S be the stable point of this construction, that is, set S_i such that $S_i = S_{i+1}$ (it exists due to a finite number of nodes in G and by the monotonicity of the recursive construction). to be the set of node Let S^* be the set of nodes v such that there is a path of length at most ℓ from s to v in the underlying network, containing nodes in $V(\mathcal{T}_{in})$ or other nodes of degree at most ψ in the network (end nodes s, v are allowed to have more than ψ neighbors in the network). Note that S^* is a union of set $S \cup V(\mathcal{T}_{in})$ and the set containing nodes v such that v is a neighbor of some node in $S \cup V(\mathcal{T}_{in})$ and there is a path in the underlying network of length at most ℓ from s to v through nodes in $S \cup V(\mathcal{T}_{in})$.

The *correctness assumption* and *correctness guarantees* for the procedure Shallow_Tide$(s, \ell, \psi, \mathcal{T})$ are defined as follows:

> A. Tree $\mathcal{T}_{in}$ is a BFS tree on a subgraph of the network G induced by the subset of nodes of distance at most m from the source s, for some $m < \ell$.
> G1. Tree $\mathcal{T}_{out}$ is a BFS tree of depth at most ℓ rooted at s in the subgraph of the underlying network induced by nodes in S^*.
> G2. The outputted value of variable $\mathcal{H}$ is the subgraph of the network containing nodes in S^* together with the edges adjacent to any of these nodes.

Shallow_Tide$(s, \ell, \psi, \mathcal{T})$ — procedure for node $v \neq s$:

Upon first arrival of message(s) to node v with status dormant:

1. Joining round:

   Receive: all point-to-point messages $(\textit{counter}', s, \ell, \psi, \mathcal{T}', \mathcal{H}')$ sent to node v in sending part of the preceding round;

   Compute: Do the following for the set of received messages containing some value *counter*$'$, unless it is bigger than ℓ:

   (a) *counter* $\leftarrow$ the smallest received *counter*$'$;

   (b) create local variable $\mathcal{T}$ by adding nodes and edges from the received copies of $\mathcal{T}'$;
   create local variable $\mathcal{H}$ by adding nodes and edges from the received copies of $\mathcal{H}'$;

   (c) $A \leftarrow N(v) \setminus V(\mathcal{H})$;

   (d) update $\mathcal{H}$ by adding nodes from $\{v\} \cup N(v)$ and edges between v and $N(v)$, if they are not in $\mathcal{H}$;

   (e) if v in $\mathcal{T}$ then $p(v) \leftarrow$ the parent of v in $\mathcal{T}$;
   else $p(v) \leftarrow$ the node with the smallest id among those from which v received a message in the current round and add node v and edge $\{v, p(v)\}$ to $\mathcal{T}$;

   (f) if *counter* $< \ell$ and v is an internal node in $\mathcal{T}$ then message $(\textit{counter} + 1, s, \ell, \psi, \mathcal{T}, \mathcal{H})$ is scheduled to be sent to all children of v in $\mathcal{T}$;

   (g) if *counter* $< \ell$ and v is *not* an internal node in $\mathcal{T}$ and $|N(v)| \leq \psi$ then message $(\textit{counter}+ 1, s, \ell, \psi, \mathcal{T}, \mathcal{H})$ is scheduled to be sent to nodes in A;

   Send: all point-to-point messages scheduled to be sent in this round;

2. Stay idle for $2 \cdot (\ell - \textit{counter})$ rounds;

3. Backward propagation round:

   Receive: all point-to-point messages $(\textit{counter}', s, \ell, \psi, \mathcal{T}', \mathcal{H}')$ sent to node v in sending part of the preceding round;

   Compute: Do the following for the set of received messages:

   (a) update local copy of $\mathcal{T}$ by adding new nodes and edges from the received copies of $\mathcal{T}'$;
   update local copy of $\mathcal{H}$ by adding new nodes and edges from the received copies of $\mathcal{H}'$;

   (b) message $(\textit{counter} + 1, s, \ell, \psi, \mathcal{T}, \mathcal{H})$ is scheduled to be sent to node $p(v)$;

   Send: point-to-point message scheduled to be sent in this round;

4. Stay idle for *counter* rounds.

Fig. 2: Procedure Shallow_Tide$(s, \ell, \psi, \mathcal{T})$: pseudo-code for node $v \neq s$.

Theorem 1 *If the initial tree $\mathcal{T}$ satisfies the correctness assumption A then*
(i) Shallow_Tide$(s, \ell, \psi, \mathcal{T})$ satisfies the correctness guarantees G1 and G2,
(ii) all participating nodes terminate in $2\ell + 2$ rounds after starting the procedure at source s, and
(iii) the number of point-to-point messages sent by nodes running the procedure is $O(|V(\mathcal{T}_{in})|+|V(\mathcal{T}_{out})\setminus V_{int}(\mathcal{T}_{in})| \cdot \psi)$, where $V_{int}(\mathcal{T}_{in})$ is the set of internal nodes in $\mathcal{T}_{in}$.

Proof: Assume that the initial tree $\mathcal{T}_{in}$ satisfies the correctness assumption A. Consider the set of nodes $V(\mathcal{T}_{out})$ of the tree outputted by the procedure. Note that the initial tree is a subgraph of the outputted tree, by step 1(e) in Figure 2. It also follows by a straightforward inductive argument based on joining rounds (step 1 in in Figure 2) that all nodes $v \in S^*$ receive a message by the end of round ℓ. Therefore, they add themselves to the tree $\mathcal{T}$ (step 1(e) in Figure 2) and to the graph $\mathcal{H}$ (step 1(d) in Figure 2; in case of $\mathcal{H}$, also the adjacent edges are added) during one of these rounds, mainly, in the first round they receive a message (joining round). The path along which the message has been propagated from s to v is the shortest one via the subgraph induced by nodes in $V(\mathcal{T}_{in}) \cup S$, by the fact that $\mathcal{T}_{in}$ is a BFS tree on a subgraph induced by some set of nodes, while node v itself is in set S^* (which contains set $V(\mathcal{T}_{in})\cup S$ and some of its adjacent nodes). It remains to note that in rounds $\ell + 1, \ldots, 2\ell$, the information is propagated backwards from such nodes v to the source. Hence it reaches the source, which updates both variables. This completes the proof of correctness guarantees G1 and G2.

Suppose that the correctness assumption is satisfied. Observe that the total number of point-to-point messages sent in an execution of the procedure Shallow_Tide$(s, \ell, \psi, \mathcal{T})$ by the participating nodes is $O(|V(\mathcal{T}_{in})| + |V(\mathcal{T}_2) \setminus V_{int}(\mathcal{T}_{in})| \cdot \psi)$. (Recall that $V_{int}(\mathcal{T}_{in})$ denotes the set of internal nodes in $\mathcal{T}_{in}$.) Indeed, the number of point-to-point messages sent by internal nodes in $\mathcal{T}_{in}$ is at most $2|V(\mathcal{T}_{in})|$, c.f., steps 1(f) and 3(b) in Figure 2. The number of messages sent by other nodes is at most $(|S^*| - |V_{int}(\mathcal{T}_{in})|) \cdot \psi + (|S^*| - |V_{int}(\mathcal{T}_{in})|) \leq 2 \cdot |V(\mathcal{T}_{out}) \setminus V_{int}(\mathcal{T}_{in})| \cdot \psi$; here the first and the second parts follow from steps 1(g) and 3(b) in Figure 2, respectively. Since $|S^*| = |V(\mathcal{T}_{out})|$, and since the number of messages sent by the source s in round 0 is at most $|V(\mathcal{T}_{out})| = |V_{int}(\mathcal{T}_{in})| + |V(\mathcal{T}_{out}) \setminus V_{int}(\mathcal{T}_{in})|$, by the correctness guarantee G1, we get the desired formula for the message complexity. □

3 BFS Algorithm

The main tool delivered in this paper is a deterministic distributed algorithm finding a BFS tree in ad hoc network with non-spontaneous wake-ups. We first give an algorithm parametrized by an additional integer ψ, which illustrates the main ideas developed in this paper. Next we analyze the correctness and complexity of this algorithm, with respect to network parameters n, D and the input parameter ψ. Finally, we extend this algorithm so that instead of independent parameter ψ there is a function f as a part of the input, which in turn gets rid of parameter ψ from the complexity formulas.

3.1 Parametrized BFS algorithm

We call our first algorithm BFS_by_Tides; Figure 3 shows the pseudo-code of BFS_by_Tides(s, ψ). There are two input parameters of this algorithm: a source s, and an integer parameter ψ. The algorithm uses the following variables:

BFS_by_Tides(s, ψ)

1. $\kappa \leftarrow 0, \mathcal{T} \leftarrow (\{s\} \cup N(s), \{\{s, v\} : v \in N(s)\})$, *termination* $\leftarrow false$;
2. Repeat
   (a) $\lambda \leftarrow 0, \mathcal{C} \leftarrow \emptyset$;
   (b) Repeat
      i. $\lambda \leftarrow \lambda + 1$;
      ii. Execute Shallow_Tide$(s, \kappa + 2^{\lambda}, \psi, \mathcal{T})$;
      let $\mathcal{H}$ stand for the outputted graph and let $\mathcal{T}$ be the outputted tree;
      iii. If $V(\mathcal{T}) = V(\mathcal{H})$ then *termination* $\leftarrow true$
      else $\mathcal{C} \leftarrow \{v : v \in V(\mathcal{H}) \setminus V_{int}(\mathcal{T}) \ \& \ |N_{\mathcal{H}}(v)| > \psi\}$;
   Until $\mathcal{C} \neq \emptyset$ or *termination*
   (c) If *termination* $= false$ then
      i. Compute locally κ: the distance from s to the nearest node in $\mathcal{C}$ counted in the graph $\mathcal{H}$;
      ii. Compute locally $\mathcal{T}$: a BFS tree of depth $\kappa + 1$ in graph $\mathcal{H}$;

   Until *termination* $= true$

OUTPUT: $\mathcal{T}$

Fig. 3: Algorithm BFS_by_Tides(s, ψ): pseudo-code for the source node s; all other nodes only participate in the runs of procedure Shallow_Tide triggered by s during the execution of BFS_by_Tides(s, ψ).

- *termination* — a boolean variable, initially set to $false$, indicating whether a certain termination condition holds; it is checked by the source node s;
- κ — a non-negative integer; its value is not decreasing, and intuitively it stores the depth until which the underlying network is *fully* known by the source s;
- λ — a non-negative integer, which, when increased by 1, doubles the range for learning new nodes and links in the network;
- $\mathcal{T}$ — a tree, monotonically non-decreasing (in the sense of inclusion), which is used as starting point to improve the estimate of the final BFS tree; initialized to node s and its neighbors with the edges connecting the neighbors with s;
- $\mathcal{H}$ — local estimate of the final network topology;
- $\mathcal{C}$ — a set of newly visited nodes during the current run of procedure Shallow_Tide, of degree bigger than ψ in the underlying network.

The output is the BFS tree $\mathcal{T}$ on the whole network, rooted at s, and graph $\mathcal{H}$ illustrating the whole topology of the underlying network. They are initialized by the source node and they are modified in the

course of the algorithm so that their final values in the termination round are returned as the output at the source node s.

The idea of the algorithm is as follows. The external loop (step 2 in the pseudo-code in Figure 3) is run until no new node is discovered, i.e., variable *termination* is set to $true$. Within a single run of this loop, an internal loop is executed (step 2(b) in the pseudo-code), which runs the procedure Shallow_Tide on subsequently increased depths $\ell = \kappa + 2^\lambda$, until the algorithm discovers a new node in $\mathcal{T}$ with degree bigger than ψ, i.e., when $C \neq \emptyset$. Then the range κ — within which the source knows the full topology of the network — is updated to be the distance to the closest of such discovered nodes (measured in the current topology-estimate graph $\mathcal{H}$), as so the estimate of the BFS tree $\mathcal{T}$ becomes up to depth $\kappa + 1$; all these changes are done in step 2(c), provided there are still unvisited nodes (i.e., $V(\mathcal{T}) \neq V(\mathcal{H})$). Figure 4 contains an illustration of three major subsequent steps of the computation.

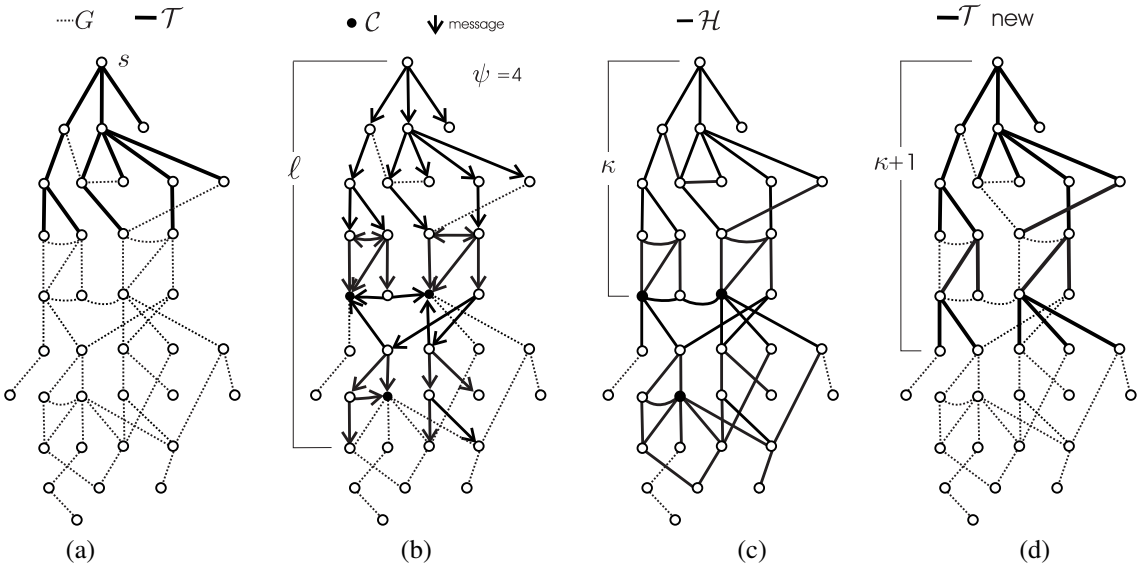

Fig. 4: An example of three steps of an execution of algorithm BFS_by_Tides: (a) tree $\mathcal{T}$ before executing procedure Shallow_Tide in step 2(b).ii of the pseudo-code; (b) messages (directed) sent during the succeeding execution of Shallow_Tide, for distance parameter $\ell = \kappa + 2^\lambda$ and tree $\mathcal{T}$ from picture (a); (c) newly learnt graph $\mathcal{H}$ with non-empty set $\mathcal{C}$ of newly discovered nodes of degree bigger than ψ (black nodes); here κ is the distance from the source s to the closest node in $\mathcal{C}$ (c.f., step 2(c).i of the pseudo-code); (d) newly defined tree $\mathcal{T}$, based on graph $\mathcal{H}$ and parameter $\kappa + 1$ (step 2(c).ii of the pseudo-code).

3.2 Analysis of algorithm BFS_by_Tides

We argue about correctness, termination and complexity of algorithm BFS_by_Tides(s, ψ), for any node s and positive integer ψ. Before starting the actual analysis, observe that the computation specified in points 2(b)iii and 2(c) is actually done by the root, and is based on the graph $\mathcal{H}$ returned by the last

execution of routine Shallow_Tide.[ii]

Correctness is guaranteed by the following properties. If the algorithm terminates, it must have $V(\mathcal{T}) = V(\mathcal{H})$ confirmed in step 2(b).iii. By Theorem 1, the execution of Shallow_Tide$(s, \kappa + 2^\lambda, \psi, \mathcal{T})$ in the preceding step 2(b).ii satisfies correctness guarantees G1 and G2, provided the correctness assumption A holds. This is assured by a straightforward induction argument on the number of executions of Shallow_Tide. By guarantee G2 and by $V(\mathcal{T}) = V(\mathcal{H})$, there are no other edges outgoing from the set of nodes $V(\mathcal{T})$ in the underlying network, and therefore the outputted $\mathcal{T}$ is a spanning tree. It is also a BFS tree on the set of (all) nodes $V(\mathcal{T})$, by the guarantee G1. By definition of κ and guarantee G2, tree T obtained in step 2(c).ii is a BFS tree of depth $\kappa + 1$ in the whole graph G. Therefore, by induction, the assumption A holds in every iteration. This completes the proof of correctness.

Termination follows from the following two observations. First, each internal loop 2(b) terminates eventually, since in each iteration parameter λ increases by one, and thus, by the correctness of the algorithm and the routine Shallow_Tide$(s, \kappa + 2^\lambda, \psi, \mathcal{T})$ (c.f., Theorem 1), the outputted tree $\mathcal{T}$ spans the whole network. This results in $V(\mathcal{T}) = V(\mathcal{H})$ and consequently *termination* being set to $true$, unless the loop has finished earlier by the condition $\mathcal{C} \neq \emptyset$ being satisfied. Moreover, after each such termination, an integer variable κ is increased by at least 1. Since κ cannot be bigger than D, as otherwise we would get $V(\mathcal{T}) = V(\mathcal{H})$ after an execution of subroutine Shallow_Tide, the number of iterations of the external loop is bounded.

Hence we proved the following.

Lemma 2 *Algorithm BFS_by_Tides(s, ψ) is correct and terminates.*

Theorem 3 *If the number of nodes in network G is n and the diameter of the network is D then*
(i) the algorithm BFS_by_Tides(s, ψ) computes a BFS tree, rooted at s, on the whole network,
(ii) all nodes terminate in $O(D \cdot \min(D, n/\psi) \cdot \log D)$ rounds after starting the algorithm at source s, and
(iii) the number of point-to-point messages sent by nodes running the algorithm is $O(n \cdot \min(D, n/\psi) \cdot \log D + n\psi \log D)$.

Theorem 3 follows from Lemma 2 and from Lemmas 5 and 6. Figure 5 illustrates an example of tree $\mathcal{T}$, graph $\mathcal{H}$, set $\mathcal{C}$, distances κ and $\ell = \kappa + 2^\lambda$ during an execution of BFS_by_Tides on network G.

For the purpose of the complexity analysis, we introduce the following terminology:

- let $\mathcal{T}_{k,i}$ be the tree $\mathcal{T}$ computed by the subroutine Shallow_Tide$(s, \kappa, \psi, \mathcal{T})$ (step 2(b).ii of the pseudo-code) when $\kappa = k$ and $\lambda = i$;
- let $\mathcal{H}_{k,i}$ be the graph $\mathcal{H}$ computed by the subroutine Shallow_Tide$(s, \kappa, \psi, \mathcal{T})$ (step 2(b).ii of the pseudo-code) when $\kappa = k$ and $\lambda = i$.

3.2.1 Time complexity

We give more accurate estimate of the time complexity of algorithm BFS_by_Tides.

Lemma 4 *The number of single runs of the external loop of algorithm BFS_by_Tides(s, ψ) is bounded above by $\min(D, 3n/\psi)$, where n is the actual number of nodes and D is the diameter of the network.*

[ii] Note that the source can easily propagate the topology of the computed BFS tree to all nodes at the costs not bigger than the costs of algorithm BFS_by_Tides.

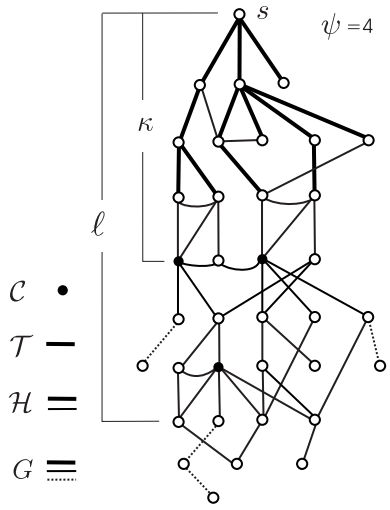

Fig. 5: An example of variables κ, $\ell = \kappa + 2^\lambda$, $\mathcal{T}$, $\mathcal{H}$ and $\mathcal{C}$ in a single round of execution BFS_by_Tides on network G at source node s. Here $\psi = 4$. Recall that $\mathcal{T}$ is a subgraph of $\mathcal{H}$ and $\mathcal{H}$ is a subgraph of G.

Proof: It is enough to estimate the number of exits from the internal loop, and more precisely, the number of different BFS levels on which a node of degree bigger than ψ is discovered. (Note that setting *termination* to true in step 2(b).iii automatically results in finishing the algorithm after the next few lines of local computation.) In one way, the number of different BFS levels in the network, starting from the source node s, containing nodes with degree bigger than ψ (in the network) is bounded naturally by network diameter D.

On the other hand, let $L_1, L_2, \ldots, L_x$ be a sequence of different BFS levels containing nodes with degree bigger than ψ in the network. Let s_j, for $1 \leq j \leq x$, be such a node in L_j of degree bigger than ψ. Observe that any node can be a neighbor of at most three such nodes s_j, mainly the one located in the same BFS layer and the other two located in the preceding and in the succeeding layers, if any. Hence, the sum $|N(s_1)| + |N(s_2)| + \ldots + |N(s_x)|$ is bounded above by $3n$. Consequently, the number x of different levels containing nodes with degree bigger than ψ is bounded from above by $3n/\psi$. □

Lemma 5 *Time complexity of algorithm BFS_by_Tides is $O(D \cdot \min(D, n/\psi) \cdot \log D)$, where n is the actual number of nodes and D is the diameter of the network.*

Proof: By Lemma 4, the number of single runs of the external loop of algorithm BFS_by_Tides(s, ψ) is bounded above by $\min(D, 3n/\psi)$.

It remains to estimate the number of rounds taken by subsequent executions of the internal loop in step 2. Note that a single run of the internal loop takes $2 \cdot (\kappa + 2^\lambda) + 2$, by Theorem 1. It follows from the structure of the internal loop, mainly from the fact that the parameter 2^λ grows exponentially and it cannot be bigger than D, that there are at most $\log D$ of executions of procedure Shallow_Tide, with different parameters, within one execution of the internal loop 2(b). Each such execution takes $O(D)$ rounds, since again the value of $2 \cdot (\kappa + 2^\lambda) + 2$ is $O(D)$ (otherwise we get $V(\mathcal{T}) = V(\mathcal{H})$ and terminate immediately).

Therefore, a single run of the external loop takes $O(D \log D)$ rounds, which combined with Lemma 4 completes the proof. □

3.2.2 Message complexity

Lemma 6 *The total number of point-to-point messages sent during the algorithm* BFS_by_Tides(s, ψ) *is* $O(n \cdot \min(D, n/\psi) \cdot \log D + n\psi \log D)$*, where* n *is the actual number of nodes and* D *is the diameter of the network.*

Proof: Partition the set of point-to-point messages sent during the execution of BFS_by_Tides(s, ψ) into two categories. First category contains messages that are sent in steps 1f and 3b of routine Shallow_Tide$(s, \ell, \psi, \mathcal{T})$ by nodes v (c.f., Figure 2), which is executed in step 2(b).ii of the pseudo-code. We call this type of messages *tree messages*. Second category consists of messages that are sent in step 1g of routine Shallow_Tide$(s, \ell, \psi, \mathcal{T})$ by nodes $v \neq s$ (c.f., Figure 2), which is executed in step 2(b).ii of the pseudo-code. We call this type of messages *searching messages*.

First we estimate the number of tree messages. By Lemma 4, the number of single runs of the external loop of algorithm BFS_by_Tides(s, ψ) is $O(\min(D, n/\psi))$. The number of iterations in the internal loop is at most $\log D$, by the upgrade rule for λ and the upper bound D on 2^λ (otherwise we would get $V(\mathcal{T}) = V(\mathcal{H})$ and terminate). Hence, the total number of executions of routine Shallow_Tide in step 2(b).ii is bounded from above by $O(\min(D, n/\psi) \cdot \log D)$. Each tree message corresponds to a single edge from tree $\mathcal{T}$ being the input of the routine Shallow_Tide during which the message was sent. Observe that in a single execution of Shallow_Tide in step 2(b).ii each edge from tree $\mathcal{T}$ is used twice: in step 1f and in step 3b. Hence, the total number of tree messages used during a single execution of routine Shallow_Tide is $O(n)$, and consequently, the total number of tree messages in the whole execution of BFS_by_Tides(s, ψ) is $O(n \cdot \min(D, n/\psi) \cdot \log D)$.

In the remainder of the proof we bound the total number of searching messages. Observe that in each step 1g of routine Shallow_Tide$(s, \ell, \psi, \mathcal{T})$ a node sends at most ψ point-to-point messages and in step 3b only one message. We show that any node v sends searching messages in at most $\log D$ rounds in the execution, i.e., at most $\log D$ times node v executes step 2(b).ii of the pseudo-code of BFS_by_Tides, triggering routine Shallow_Tide. This will automatically imply the upper bound $O(n \cdot \psi \cdot \log D)$ for the total number of searching messages.

Note that node s does not send searching messages, as tree $\mathcal{T}$ always contains all its adjacent edges in network G. Consider node $v \neq s$. Let k_j, i_j, for a positive integer j, be the values of variables κ, λ, respectively, when v sends searching messages for the j-th time during the execution of BFS_by_Tides(s, ψ). More precisely, each such j-th sending of searching messages by node v happens when it participates in the execution of the routine Shallow_Tide$(s, k_j + 2^{i_j}, \psi, \mathcal{T})$ in step 2(b).ii of the pseudo-code of BFS_by_Tides(s, ψ), when it runs its step 1g of the routine. Observe that there is at most one run of step 1g by a node in any execution of routine Shallow_Tide. Moreover, only nodes that are not internal in the input instance of $\mathcal{T}$ may send any searching message within routine Shallow_Tide.

We first argue that the sequence $(k_j)_j$ is increasing. Indeed, it is non-decreasing by the fact that the value of variable κ is non-decreasing in time. Suppose, to the contrary, that $k_j = k_{j+1}$, for some i. It follows that $1 \leq i_j < i_{j+1} \leq D$. Moreover, the routine Shallow_Tide$(s, k_{j+1} + 2^{i_{j+1}}, \psi, \mathcal{T}_{k_{j+1}, i_{j+1}})$ is executed after Shallow_Tide$(s, k_j + 2^{i_j}, \psi, \mathcal{T}_{k_j, i_j})$ but within the same execution of step 2(b) of the pseudo-code, and therefore tree $\mathcal{T}_{k_{j+1}, i_{j+1}}$ contains the tree outputted by the routine Shallow_Tide$(s, k_j + 2^{i_j}, \psi, \mathcal{T}_{k_j, i_j})$. In particular, $\mathcal{T}_{k_{j+1}, i_{j+1}}$ contains node v as internal node, by specification of step 1g

(mainly, the fact that *counter* $< \ell = k_j + 2^{i_j}$) run during the execution of Shallow_Tide$(s, k_j + 2^{i_j}, \psi, \mathcal{T}_{k_j,i_j})$. It follows that v cannot send a searching message in the following executions of Shallow_Tide for parameter $\kappa = k_{j+1}$, in particular for $\lambda = i_{j+1}$, as it is an internal node in their input instances of tree $\mathcal{T}$; thus we get a contradiction, which completes the proof that the sequence $(k_j)_j$ is increasing.

It remains to argue that the sequence $(i_j)_j$ is decreasing and within the range $[1, \log D]$. The range follows from the same argument as given in the estimate of the number of tree messages. Suppose that $i_j = i_{j+1}$, for some j. We already argued that $k_j < k_{j+1}$. It follows from the fact that set $\mathcal{C}$ was empty after the execution of Shallow_Tide$(s, k_j + 2^{i_j-1}, \psi, \mathcal{T}_{k_j,i_j-1})$ that $k_{j+1} > k_j + 2^{i_j-1}$. It follows that $k_{j+1} + 2^{i_{j+1}-1} = k_{j+1} + 2^{i_j-1} > k_j + 2^{i_j-1} + 2^{i_j-1} = k_j + 2^{i_j}$. On the other hand, the shortest path from s to v in network G is at most $k_j + 2^{i_j}$, since v participates in the execution of Shallow_Tide$(s, k_j + 2^{i_j-1}, \psi, \mathcal{T}_{k_j,i_j-1})$. These two facts together yield that node v must have been added to set $\mathcal{T}$ as an internal node during Shallow_Tide$(s, k_j + 2^{i_j-1}, \psi, \mathcal{T}_{k_j,i_j-1})$, and consequently it could not send any message during Shallow_Tide$(s, k_j + 2^{i_j-1}, \psi, \mathcal{T}_{k_j,i_j-1})$. We obtained a contradiction, proving that the sequence $(i_j)_j$ is decreasing and within the range $[1, \log D]$. Hence the number of rounds in which node v sends searching messages is at most $\log D$. □

4 BFS algorithm with complexity formula being a function of n and D only

The complexity of algorithm BFS_by_Tides(s, ψ) depends on the input parameter ψ, which is independent of the actual number of nodes n. We show how to modify algorithm BFS_by_Tides and then how to iterate the obtained algorithm to adapt to the number n, in the sense that the complexity of the obtained solution will be expressed only as a function on network parameters n, D. Recall that these parameters are not a priori known to the nodes, therefore the resulting algorithm must estimate their values during the execution and act accordingly.

Algorithm Bounded_BFS. We first show how to modify algorithm BFS_by_Tides according to our needs of further elimination of parameter ψ. This modification still uses ψ as parameter, however it helps to control the number of discovered nodes and terminate as soon as this number becomes bigger than some input parameter n^*. We call the modified algorithm Bounded_BFS, and run with three input parameters: s, ψ, n^*. The pseudo-code of the algorithm Bounded_BFS(s, ψ, n^*) differs from the pseudo-code of BFS_by_Tides(s, ψ) in two details. The first difference is in executing the following step instead of step 2(b).iii :

2(b).iii* : If $V(\mathcal{T}) = V(\mathcal{H})$ or $|V(\mathcal{H})| > n^*$ then *termination* $\leftarrow true$
else $\mathcal{C} \leftarrow \{v : v \in V(\mathcal{H}) \setminus V_{int}(\mathcal{T}) \ \&\ |N_{\mathcal{H}}(v)| > \psi\}$;

It can be easily observed that the only difference between the original step 2(b).iii of algorithm BFS_by_Tides(s, ψ) and its counterpart (the above step 2(b).iii*) in algorithm Bounded_BFS(s, ψ, n^*) is the additional alternative condition $|V(\mathcal{H})| > n^*$ yielding termination (i.e., setting variable *termination* to $true$). The second difference between the two algorithms is that in Bounded_BFS we also output the topology-estimate graph $\mathcal{H}$, as it can be further useful in the design of our final BFS algorithm.
We require from Bounded_BFS(s, ψ, n^*) to satisfy the following conditional correctness guarantee:

Correctness guarantee for Bounded_BFS(s, ψ, n^*):
If $n \leq n^*$ then the outputted $\mathcal{H}$ is the same as the underlying network topology G and the outputted $\mathcal{T}$ is a BFS tree on G rooted at s.

Theorem 7 *If the number of nodes in network G is n and the diameter of the network is D then*
(i) the algorithm Bounded_BFS(s, ψ, n^*) satisfies the correctness guarantee,
(ii) all nodes terminate in $O(D \cdot \min(D, n^*/\psi) \cdot \log D)$ rounds after starting the algorithm at source s, and
(iii) the number of point-to-point messages sent in any execution by nodes running the algorithm is $O(n \cdot \min(D, n/\psi) \cdot \log D + n\psi \log D)$.

Proof: The correctness guarantee holds by Theorem 3, as in case $n \leq n^*$ the executions of Bounded_BFS(s, ψ, n^*) and BFS_by_Tides(s, ψ) are the same when run on the same network. This is because the condition $|V(\mathcal{H})| > n^*$ in step 2(b).iii* of Bounded_BFS(s, ψ, n^*) is never satisfied, and thus the computed estimate $\mathcal{H}$ contains all nodes, and thus all their adjacent edges, while $\mathcal{T}$ is a BFS tree on this graph rooted at s.

Unlike in the limited correctness guarantee, we estimate the complexity of Bounded_BFS(s, ψ, n^*) for *any* execution on any network in the considered model, that is, even for executions that do not satisfy the condition $n \leq n^*$. This is because we will later apply Bounded_BFS(s, ψ, n^*) to build a BFS tree in any network, and thus we will need to argue about the complexity of used subroutine Bounded_BFS in any execution for any network.

Time complexity of Bounded_BFS(s, ψ, n^*) is $O(D \cdot \min(D, n^*/\psi) \cdot \log D)$. To prove it, first note that the execution of Bounded_BFS(s, ψ, n^*) up to the last checking of the condition in step 2(b).iii*, and thus up to the very last round, is the same as the corresponding prefix of the execution of BFS_by_Tides(s, ψ) in the same network (note that the only difference between the two pseudo-codes — the condition $|V(\mathcal{H})| > n^*$ — does not hold in the considered period). By Lemma 5 applied to this prefix and by the fact that the time of the last execution of routine Shallow_Tide within BFS_by_Tides is $O(D)$, we get that the time complexity of Bounded_BFS(s, ψ, n^*) is

$$O(D \cdot \min(D, n'/\psi) \cdot \log D) + O(D)$$

$$= O(D \cdot \min(D, n^*/\psi) \cdot \log D) \ ,$$

where $n' \leq n^*$ is the number of nodes in variable $\mathcal{H}$ before the last execution of routine Shallow_Tide.

We argue that the message complexity of Bounded_BFS(s, ψ, n^*) is $O(n \cdot \min(D, n/\psi) \cdot \log D + n\psi \log D)$. Similarly as for the time complexity analysis, we argue that the message complexity is the same as in BFS_by_Tides(s, ψ) until the beginning of the last execution of Shallow_Tide in Bounded_BFS(s, ψ, n^*), that is, $O(n' \cdot \min(D, n'/\psi) \cdot \log D + n'\psi \log D)$ for some $n' \leq n^*$ being the number of nodes in variable $\mathcal{H}$ before the last execution of routine Shallow_Tide. The message cost of the last execution of routine Shallow_Tide in the execution of Bounded_BFS(s, ψ, n^*) is $O(n \cdot \psi)$, by Theorem 1. □

Main algorithm Final_BFS. Our main algorithm Final_BFS(s, f), where f is a non-decresing integer function, is presented in Figure 6. The algorithm uses variable *Nnodes* to estimate the upper bound on the number of nodes in the network; it is initiated to $|N(s)| + 1$ and updated to $2^{\lceil \log_2 |V(\mathcal{H})| \rceil}$ after each

Final_BFS(s, f)

1. *Nnodes* $\leftarrow |N(s)| + 1$;
2. Repeat
   (a) Execute Bounded_BFS$(s, f(\mathit{Nnodes}), \mathit{Nnodes})$;
   (b) *Nnodes* $\leftarrow 2^{\lceil \log_2 |V(\mathcal{H})| \rceil}$;

   Until $V(\mathcal{T}) = V(\mathcal{H})$

OUTPUT: $\mathcal{T}$

Fig. 6: Algorithm Final_BFS(s, f): pseudo-code for the source node s; all other nodes only participate in the executions of Bounded_BFS triggered by s during the execution of Final_BFS(s, f).

execution of routine Bounded_BFS$(s, f(\mathit{Nnodes}), \mathit{Nnodes})$. Recall that routine Bounded_BFS outputs tree $\mathcal{T}$ and an estimate of network topology $\mathcal{H}$. This is repeated until the sets of nodes in outputted $\mathcal{T}$ and $\mathcal{H}$ are equal, i.e., there is no edge adjacent to any node in $V(\mathcal{T})$ that leads to a node that is outside of $V(\mathcal{T})$.

Theorem 8 *If f is an integer function such that $\frac{x}{f(x)}$ is a non-decreasing function of x, then for any network of n nodes and diameter D*
(i) the algorithm Final_BFS(s, f) computes a BFS tree, rooted at s, on the whole network,
(ii) all nodes terminate in $O(D \cdot \min(D, n/f(n)) \cdot \log D \cdot \log n)$ rounds after starting the algorithm at source s, and
(iii) the number of point-to-point messages sent by nodes running the algorithm is $O(n \cdot (\min(D, n/f(n)) + f(n)) \cdot \log D \cdot \log n)$.

Proof: The correctness of algorithm Final_BFS, i.e., that it outputs a BFS tree on the whole network and rooted at s, follows from the observation that when the condition $V(\mathcal{T}) = V(\mathcal{H})$ holds, there is no edge adjacent to any node in $\mathcal{T}$ that leads to a node outside of set $V(\mathcal{T})$. This is because $\mathcal{H}$ contains all nodes in $V(\mathcal{T})$ together with all their adjacent edges and their end nodes.

Termination of algorithm Final_BFS follows from the fact that *Nnodes* is an integer and it increases by at least one in each execution of step 2(b). Therefore, unless finished earlier, in some round the value of *Nnodes* will be at least n — the number of nodes in the network — and the next execution of routine Bounded_BFS will return a BFS tree on the whole network (c.f., correctness guarantee for Bounded_BFS), which in turn makes the stopping condition $V(\mathcal{T}) = V(\mathcal{H})$ fulfilled.

In order to make more accurate estimate of time complexity, observe that the number of executions of routine Bounded_BFS is in fact $O(\log n)$, since each time a single iteration of the loop finishes without termination, we have $|V(\mathcal{H})| > |V(\mathcal{T})|$, and moreover, $|V(\mathcal{H})| > \mathit{Nnodes}$, as otherwise the routine Bounded_BFS would return a BFS tree on the whole network and the algorithm Final_BFS would terminate. It follows that *Nnodes* will be at least doubled in step 2(b), which means that after $O(\log n)$ iterations it will be at least n and we could apply the same argument as in proving termination to argue about stopping by the end of this iteration. It follows from the time complexity of algo-

rithm Bounded_BFS$(s, f(n^*), n^*)$, where n^* is a power of 2, that the time complexity of algorithm Final_BFS(s, f) is at most

$$O\left(\sum_{j=0}^{\log n}\left(D \cdot \min(D, 2^j/f(2^j)) \cdot \log D\right)\right),$$

which is

$$O(D \cdot \min(D, n/f(n)) \cdot \log D \cdot \log n)$$

in case of $\frac{x}{f(x)}$ being a non-decreasing function of x. [(iii)]

Message complexity of algorithm Final_BFS is

$$O\left((n \cdot \min(D, n/f(n)) \cdot \log D + nf(n) \log D) \cdot \log n\right)$$

by the similar argument as used in the analysis of time complexity of Final_BFS and by the fact that $x/f(x)$ is a non-decreasing function. □

5 Conclusion

We achieved $o(n)$ time complexity and $o(n^2)$ message complexity for the problem of spanning a BFS tree in ad hoc networks with diameter $D = D(n) = o(n/\log^4 n)$, e.g., by setting $f(n) = \sqrt{nD(n)}$. Our technique is even more general, and establishes a tradeoff between time and message complexity functions. There is still a gap between our time-message complexity tradeoff and natural lower bound D on time and $\Omega(n)$ on message complexity. As far as we know these are the best known lower bounds on the corresponding complexity measures when considering them separately, but it is not clear whether similar bounds can be matched simultaneously by a single algorithm. Recall that the former bound is matched by a naive flooding, in the cost of $\Omega(n^2)$ messages, while the latter bound is matched by a DFS-like search algorithm collecting the whole knowledge about the underlying graph in linear time $\Theta(n)$ (independent on D). Our final deterministic distributed solution is close to the lower bounds by factors roughly $O(f(n)$ polylog $n)$ and $O(n/f(n)$ polylog $n)$, for some suitable functions $f(\cdot)$.

Acknowledgments

We are grateful to anonymous reviewers, whose comments helped to improve the overall quality of the paper.

[(iii)] For some functions f, e.g., for $f(x) = x^{1-\epsilon}$ where $\epsilon \in (0, 1)$, the $\log n$ factor can be removed from the above formula, or at least decreased.

References

B. Awerbuch. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, STOC '87, pages 230–240, New York, NY, USA, 1987. ACM. ISBN 0-89791-221-7. doi: 10.1145/28395.28421. URL http://doi.acm.org/10.1145/28395.28421.

B. Awerbuch and R. G. Gallager. Distributed bfs algorithms. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science*, SFCS '85, pages 250–256, Washington, DC, USA, 1985. IEEE Computer Society. ISBN 0-8186-0844-4. doi: 10.1109/SFCS.1985.20. URL http://dx.doi.org/10.1109/SFCS.1985.20.

B. Awerbuch and R. G. Gallager. A new distributed algorithm to find breadth first search trees. *IEEE Trans. Inf. Theor.*, 33(3):315–322, May 1987. ISSN 0018-9448. doi: 10.1109/TIT.1987.1057314. URL http://dx.doi.org/10.1109/TIT.1987.1057314.

B. Awerbuch, O. Goldreich, R. Vainish, and D. Peleg. A trade-off between information and communication in broadcast protocols. *J. ACM*, 37(2):238–256, Apr. 1990. ISSN 0004-5411. doi: 10.1145/77600.77618. URL http://doi.acm.org/10.1145/77600.77618.

B. Awerbuch, I. Cidon, and S. Kutten. Optimal maintenance of a spanning tree. *J. ACM*, 55(4):18:1–18:45, Sept. 2008. ISSN 0004-5411. doi: 10.1145/1391289.1391292. URL http://doi.acm.org/10.1145/1391289.1391292.

M. Bienkowski, L. Gasieniec, M. Klonowski, M. Korzeniowski, and S. Schmid. Event extent estimation. In *Proceedings of the 17th international conference on Structural Information and Communication Complexity*, SIROCCO'10, pages 57–71, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-13283-9, 978-3-642-13283-4. doi: 10.1007/978-3-642-13284-1_6. URL http://dx.doi.org/10.1007/978-3-642-13284-1_6.

S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Randomized gossip algorithms. *IEEE/ACM Trans. Netw.*, 14(SI): 2508–2530, June 2006. ISSN 1063-6692. doi: 10.1109/TIT.2006.874516. URL http://dx.doi.org/10.1109/TIT.2006.874516.

B. S. Chlebus and D. R. Kowalski. Robust gossiping with an application to consensus. *J. Comput. Syst. Sci.*, 72(8): 1262–1281, Dec. 2006. ISSN 0022-0000. doi: 10.1016/j.jcss.2006.08.001. URL http://dx.doi.org/10.1016/j.jcss.2006.08.001.

M. Chrobak, L. Gąsieniec, and D. R. Kowalski. The wake-up problem in multihop radio networks. *SIAM J. Comput.*, 36(5):1453–1471, Dec. 2006. ISSN 0097-5397. doi: 10.1137/S0097539704442726. URL http://dx.doi.org/10.1137/S0097539704442726.

A. E. F. Clementi, A. Monti, and R. Silvestri. Selective families, superimposed codes, and broadcasting on unknown radio networks. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, SODA '01, pages 709–718, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics. ISBN 0-89871-490-7. URL http://dl.acm.org/citation.cfm?id=365411.365756.

A. Demers, D. Greene, C. Houser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. *SIGOPS Oper. Syst. Rev.*, 22(1):8–32, Jan. 1988. ISSN 0163-5980. doi: 10.1145/43921.43922. URL http://doi.acm.org/10.1145/43921.43922.

S. M. Hedetniemi, S. T. Hedetniemi, and A. L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18(4):319–349, 1988. ISSN 1097-0037. doi: 10.1002/net.3230180406. URL http://dx.doi.org/10.1002/net.3230180406.

J. Hromkovic, R. Klasing, A. Pelc, P. Ruzicka, and W. Unger. *Dissemination of Information in Communication Networks: Broadcasting, Gossiping, Leader Election, and Fault-Tolerance (Texts in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005. ISBN 3540008462.

D. Kempe, J. Kleinberg, and A. Demers. Spatial gossip and resource location protocols. *J. ACM*, 51(6):943–967, Nov. 2004. ISSN 0004-5411. doi: 10.1145/1039488.1039491. URL http://doi.acm.org/10.1145/1039488.1039491.

E. Korach, S. Moran, and S. Zaks. Optimal lower bounds for some distributed algorithms for a complete network of processors. *Theor. Comput. Sci.*, 64(1):125–132, Apr. 1989. ISSN 0304-3975. doi: 10.1016/0304-3975(89)90103-5. URL http://dx.doi.org/10.1016/0304-3975(89)90103-5.

A. Pelc. Fault-tolerant broadcasting and gossiping in communication networks. *Networks*, 28(3):143–156, 1996. ISSN 1097-0037. doi: 10.1002/(SICI)1097-0037(199610)28:3⟨143::AID-NET3⟩3.0.CO;2-N. URL http://dx.doi.org/10.1002/(SICI)1097-0037(199610)28:3<143::AID-NET3>3.0.CO;2-N.

D. Peleg. *Distributed computing: a locality-sensitive approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000. ISBN 0-89871-464-8.