Chapter 7

All-to-All Communication

In the previous chapters, we have always considered communication on a particular graph $G = (V, E)$, where any two nodes u and v can only communicate directly if $\{u, v\} \in E$. This is however not always the best way to model a network. In the Internet, for example, every machine (node) is able to "directly" communicate with every other machine via a series of routers. If every node in a network can communicate directly with all other nodes, many problems can be solved easily. For example, assume we have n servers, each hosting an arbitrary number of (numeric) elements. If all servers are interested in obtaining the maximum of all elements, all servers can simultaneously, i.e., in one communication round, send their local maximum element to all other servers. Once these maxima are received, each server knows the global maximum.

Note that we can again use graph theory to model this *all-to-all* communication scenario: The communication graph is simply the complete graph $\mathcal{K}_n := (V, \binom{V}{2})$. If each node can send its entire local state in a single message, then all problems could be solved in 1 communication round in this model! Since allowing unbounded messages is not realistic in most practical scenarios, we restrict the message size: Assuming that all node identifiers and all other variables in the system (such as the numeric elements in the example above) can be described using $O(\log n)$ bits, each node can only send a message of size $O(\log n)$ bits to all other nodes. In other words, only a constant number of identifiers (and elements) can be packed into a single message. Thus, in this model, the limiting factor is the amount of information that can be transmitted in a fixed amount of time. This is fundamentally different from the model we studied before where nodes are restricted to local information about the network graph.

In this chapter we study one particular problem in this model, the computation of a minimum spanning tree (MST), i.e., we will again look at construction of a basic network structure. Let us first review the definition of a minimum spanning tree from Chapter 3. We assume that each edge e is assigned a weight ω_e.

Definition 7.1 (MST)**.** *Given a weighted graph $G = (V, E, \omega)$. The MST of G is a spanning tree T minimizing $\omega(T)$, where $\omega(G') = \sum_{e \in G'} \omega_e$ for any subgraph $G' \subseteq G$.*

Upper Bounds

Graph Class	Time Complexity	Authors
General Graphs	$O(D + \sqrt{n} \cdot \log^* n)$	Kutten, Peleg
Diameter 2	$O(\log n)$	Lotker, Patt-Shamir, Peleg
Diameter 1	$O(\log \log n)$	Lotker, Patt-Shamir, Pavlov, Peleg

Lower Bounds

Graph Class	Time Complexity	Authors
Diameter $\Omega(\log n)$	$\Omega(D + \sqrt{n}/\log^2 n)$	Peleg, Rubinovich
Diameter 4	$\Omega(n^{1/4}/\sqrt{\log n})$	Lotker, Patt-Shamir, Peleg
Diameter 3	$\Omega(n^{1/3}/\log n)$	Lotker, Patt-Shamir, Peleg

Table 7.1: Time complexity of distributed MST construction

Remark:

- Since we have a complete communication graph, the graph has $\binom{n}{2}$ edges in the beginning.
- As in Chapter 3, we assume that no two edges of the graph have the same weight. Recall that this makes the MST unique. Recall also that this simplification is not essential as one can always break ties by adding the IDs of adjacent vertices to the weight.

For simplicity, we assume that we have a synchronous model (as we are only interested in the time complexity, our algorithm can be made asynchronous using synchronizer α at no additional cost (cf. Chapter 6). As usual, in every round, every node can send a (potentially different) message to each of its neighbors. In particular, note that the message delay is 1 for every edge e independent of the weight ω_e. As mentioned before, every message can contain a constant number of node IDs and edge weights (and $O(\log n)$ additional bits).

There is a considerable amount of work on distributed MST construction. Table 7.1 lists the most important results for various network diameters D. As we have a complete communication network in our model, we focus only on $D = 1$.

Remarks:

- Note that for graphs of arbitrary diameter D, if there are no bounds on the number of messages sent, on the message size, and on the amount of local computations, there is a straightforward generic algorithm to compute an MST in time D: In every round, every node sends its complete state to

all its neighbors. After D rounds, every node knows the whole graph and can compute any graph structure locally without further communicating.

- In general, the diameter D is also an obvious lower bound for the time needed to compute an MST. In a weighted ring, e.g., it takes time D to find the heaviest edge. In fact, on the ring, time D is required to compute any spanning tree.

In this chapter, we are not concerned with lower bounds, we want to give an algorithm that computes the MST as quickly as possible instead! We again use the following lemma that is proven in Chapter 3.

Lemma 7.2. *For a given graph G let T be an MST, and let $T' \subseteq T$ be a subgraph (also known as a fragment) of the MST. Edge $e = (u, v)$ is an outgoing edge of T' if $u \in T'$ and $v \notin T'$ (or vice versa). Let the minimum weight outgoing edge of the fragment T' be the so-called blue edge $b(T')$. Then $T' \cup b(T') \subseteq T$.*

Lemma leads to a straightforward distributed MST algorithm. We start with an empty graph, i.e., every node is a fragment of the MST. The algorithm consists of phases. In every phase, we add the blue edge $b(T')$ of every existing fragment T' to the MST. Algorithm 26 shows how the described simple MST construction can be carried out on a network of diameter 1.

Algorithm 26 Simple MST Construction (at node v)

```
1: // all nodes always know all current MST edges and thus all MST fragments
2: while v has neighbor u in different fragment do
3:    find lowest-weight edge e between v and a node u in a different fragment
4:    send e to all nodes
5:    determine blue edges of all fragments
6:    add blue edges of all fragments to MST, update fragments
7: end while
```

Theorem 7.3. *On a complete graph, Algorithm 26 computes an MST in time $O(\log n)$.*

Proof. The algorithm is correct because of Lemma 7. Every node only needs to send a single message to all its neighbors in every phase (line 4). All other computations can be done locally without sending other messages. In particular, the blue edge of a given fragment is the lightest edge sent by any node of that fragment. Because every node always knows the current MST (and all current fragments), lines 5 and 6 can be performed locally.

In every phase, every fragment connects to at least one other fragment. The minimum fragment size therefore at least doubles in every phase. Thus, the number of phases is at most $\log_2 n$. □

Remark:

- Algorithm 26 does essentially the same thing as the GHS algorithm (Algorithm 15) discussed in Chapter 3. Because we now have a complete graph and thus every node can communicate with every other node, things become simpler (and also much faster).

- Algorithm 26 does not make use of the fact that a node can send different messages to different nodes. Making use of this possibility will allow us to significantly improve the running time of the algorithm.

Our goal is now to improve Algorithm 26. We assume that every node has a unique identifier. By sending its own identifier to all other nodes, every node knows the identifiers of all other nodes after one round. Let $\ell(F)$ be the node with the smallest identifier in fragment F. We call $\ell(F)$ the leader of fragment F. In order to improve the running time of Algorithm 26, we need to be able to connect every fragment to more than one other fragment in a single phase. Algorithm 27 shows how a fragment of size k can be connected to at least k other fragments in every phase.

Algorithm 27 Fast MST construction (at node v)

```
1:  // all nodes always know all current MST edges and thus all MST fragments
2:  repeat
3:    F := fragment of v;
4:    ∀F' ≠ F, compute min-weight edge e_F' connecting v to F'.
5:    ∀F' ≠ F, send e_F' to ℓ(F')
6:    if v = ℓ(F) then
7:      ∀F' ≠ F, determine min-weight edge e_{F,F'} between F and F'
8:      k := |F|
9:      E(F) := k lightest edges among e_{F,F'} for F' ≠ F
10:     send edges in E(F) to different nodes in F
                    // for simplicity assume that v also sends an edge to itself
11:   end if
12:   send edge received from ℓ(F) to all nodes
13:   // the following operations are performed locally by each node
14:   E' := edges received by other nodes
15:   while E' ≠ ∅ do
16:     e := lightest edge from E'
17:     E' := E' \ {e}
18:     if e does not close cycle in MST then add e to MST end if
19:   end while
20: until all nodes are in the same fragment
```

Remark:

- Note that the set of received edges E' in line 14 is the same for all nodes. Because all nodes know all current fragments, the following while loop can be computed locally by each node.

Lemma 7.4. *Whenever an edge e connecting fragments F and F' is not added to the MST in line 18, fragments F and F' are already connected by some path.*

Proof. Assume that $e = (u, v)$ for $u \in F$ and $v \in F'$. Because e closes a cycle, there already is a path connecting u and v in the part of the MST constructed so far. Because all nodes inside fragment F and all nodes inside F' are already connected by the definition of a fragment, the lemma follows. □

Lemma 7.5. *The algorithm is correct, that is, it only adds MST edges.*

Proof. Consider the while loop in lines 15–19 of Algorithm 27. Note that this is the only place where new edges are added to the MST. We want to prove that at the time we add an edge e in line 18, e is the blue edge of some fragment F. The lemma then follows from Lemma 7. Edge e is an edge from the set $E(F)$ of some fragment F. By Lemma 7.4, at the time e is added, F is connected to all fragments F' for which there is an edge $e_{F,F'}$ connecting F and F' such that $\omega_{e_{F,F'}} < \omega_e$. Furthermore, e is the lightest edge connecting F to F'. Hence, e is the blue edge of F. □

Theorem 7.6. *Algorithm 27 computes an MST in time $O(\log \log n)$.*

Proof. Lemma 7.5 shows that the algorithm computes an MST. We therefore only have to prove that the algorithm terminates in $O(\log \log n)$ rounds. Clearly, every phase only needs a constant number of rounds. It thus remains to show that the number of phases is at most $O(\log \log n)$. Assume that k is the minimum fragment size at the beginning of a phase. By Lemma 7.4, a fragment F is connected to at least $|E(F)| = |F|$ other fragments at the end of the phase. Because all fragments have size at least k every fragment is connected to at least k other fragments of size at least k. The minimum fragment size hence grows to at least $k(k+1)$. After the first phase, fragments have size at least 2. From then on, the minimum fragment size is at least the square of the minimum fragment size of the previous phase. Thus, the minimum fragment size after $f+1$ phases is at least 2^{2^f} (squaring doubles the exponent in every phase). Therefore after $1 + \log_2 \log_2 n$ phases, the minimum fragment size is n and thus, all nodes are in the same fragment. □

Remarks:

- It is not known whether the $O(\log \log n)$ time complexity of Algorithm 27 is optimal. In fact, no lower bounds are known for the MST construction on diameter 1 and 2 graphs.
- Algorithm 27 makes use of the fact that it is possible to send different messages to different nodes. If we assume that every node always has to send the same message to all other nodes, Algorithm 26 is the best that is known. Also for this simpler case, no lower bound is known.