The system

Asynchronous Message passing systems:

- a graph *G(V,E)*, called *communication graph*, where nodes in *V* represent processes and edges *(u,v)* in *E* represent a link from *u* to *v*.

Through a link *(u,v)* process *u* can send messages to process *v*. Each process knows its state and can register following events:

- *send event*: a message is produced i.e it is sent to another process
- *receive event:* a message that was sent on a link is consumed by a process
- *internal event:* local computation

On an event a process performs a transition which means that it changes its state and might trigger some new events.

Distributed Algorithms and its Quality

The quality of correct algorithms is analysed by their complexity measures. In order to measure the complexity of a distributed algorithm in a message passing system one usually considers the TIME COMPLEXITY and the COMMUNICATION COMPLEXITY.

time COMPLEXITY measures the *maximum time* the algorithm needs in the *worst case* starting with its initialisation until it comes to a halt. For this purpose an idealised timing is used in which the time is measured in message transmission units.

Definition 1. *it is assumed that the transmission of a message takes at most one time unit and the time which a process task needs to proceed is encapsulated in the time unit.*

The time complexity measures the maximum time the algorithm needs in the worst case starting with its initialisation until it comes to a halt.

Complexities

communication COMPLEXITY measures the traffic load of the system.

This is achieved by counting the TOTAL NUMBER OF MESSAGES that are EXCHANGED among the system in the WORST CASE. Also the size of messages might be interesting if it is not constant.

Broadcast

A BROADCAST ALGORITHM distributes an information known by a single process to all other processes of the network.

A BROADCAST ALGORITHM is a basic algorithm used inside many other distributed algorithms e.g during initialisation of a network where all processes should be waken up.

A strait forward BROADCAST ALGORITHM:

- The initiator sends to all its neighbours a message of kind *broadcast*.
- When a process receives message *broadcast* for the first time, it sends to all other adjacent processes further *broadcast* messages.

BC-Complexities

Time *Complexity: $O(D)$. D* denotes the diameter of the *Communication Graph*.

Proof. After ONE time unit the distance between the initiator and the processes which have not received the broadcasted information decreases at least by one. $\square$

Message *Complexity: $O(|E|)$.*

Proof. Every process needs to send at most one message on a edge. $\square$

BC with Acknowledgement

Often the process initiating the broadcast algorithm needs to be acknowledged that all processes received the broadcasted information. Algorithms satisfying this property are called BROADCAST WITH ACKNOWLEDGEMENT algorithms.

A BROADCAST WITH ACKNOWLEDGEMENT ALGORITHM:

Extend the BC algorithm.

- As before the initiator sends messages of kind *broadcast* to all its neighbours.
- Also processes that receive message *broadcast* for the first time still send to all other adjacent processes further *broadcast* messages.

In addition

- a process marks the sender of the first received *broadcast* message as its *parent*.
- Processes which receive a message *broadcast* although they have already decided for their parent node reply with an *acknowledgement*.
- every process keeps track on *broadcast* messages it sent to adjacent processes by storing the receiver of each message in a SET OF EXPECTED ACKNOWLEDGEMENTS, called *ack*.
- As long as *ack* is not empty the process waits to receive *acknowledgements* from processes stored in this set. On receiving an *acknowledgement* a process deletes the sender from *ack*.
- When *ack* is an empty set then all *acknowledgements* have been received. Then a process sends an

acknowledgement to its *parent node* (if it has a *parent node*) and exits the algorithm.

Why does it work?

- It broadcasts: all processors will receive the *broadcast message.*

- It terminates: every *broadcast* message will be answered by an acknowledgement.

- It terminates correctly. (?)

It terminates correctly

The algorithm computes a SPANNING TREE which is formed by the edges on which a process receives its first *broadcast* message.

Proof. When a processor receives all *acks* it knows that all it's descendants in the SPANNING TREE have received the information. (proof by induction starting from the leaves of the tree). □

Message Complexity: *O(|E|)*.

Proof. Every process sends along each adjacent link at most one *broadcast message* and one *ack* message. □

Time Complexity:

- **Phase 1** According to previous algorithm every process received the broadcasted information in time *O(D)* and has computed a spanning tree of DEPTH $d <= n$.

- **Phase 2** In the convergecast phase of the algorithm the distance between the initiator and processes having received all acknowledgements decreases by at least one after time 1, so it takes *O(n)* until the initiator has received all acknowledgements.

- Hence, the whole algorithm is guaranteed to terminate in time *O(n)*.

Spanning Tree and later Broadcasts

This spanning tree can be useful for further broadcasts. Broadcast only on the spanning tree.

Then each process receives at most one *broadcast* message and sends one *ack*.

Message complexity is reduced to $O(n)$.

Do we need a different spanning tree for every initiator?