1.2 Modeling Asynchronous Dynamic Links

We now specify how communication is assumed to occur over the dynamic links, and how notification of a links status is synchronized at the two endpoints of the link.

The state of a link Linku, v, which models the bidirectional communication link between node u and node v, consists of a status variable and two queues of messages.

The possible values of the status variable are Up, $GoingDown_u$, $GoingDown_v$, $Down$, $ComingUp_u$, and $ComingUp_v$. The link transitions among different values of its status variable through $LinkUp$ and $LinkDown$ events. Figure 1 shows the state transition diagram for $Link_{u,v}$. The intuition is that if a LinkUp (resp., LinkDown) occurs at one endpoint of the link, then LinkUp (resp., LinkDown) must occur at the other endpoint before LinkDown (resp., LinkUp) can occur at either end.

The other components of the links local state are the two message queues: mqueue u,v holds messages in transit from u to v and mqueue v,u holds messages in transit from v to u.

An attempt by node u to send a message to node v results in the message being appended to mqueue u,v if the links status is either ComingUp u or Up; otherwise there is no effect.

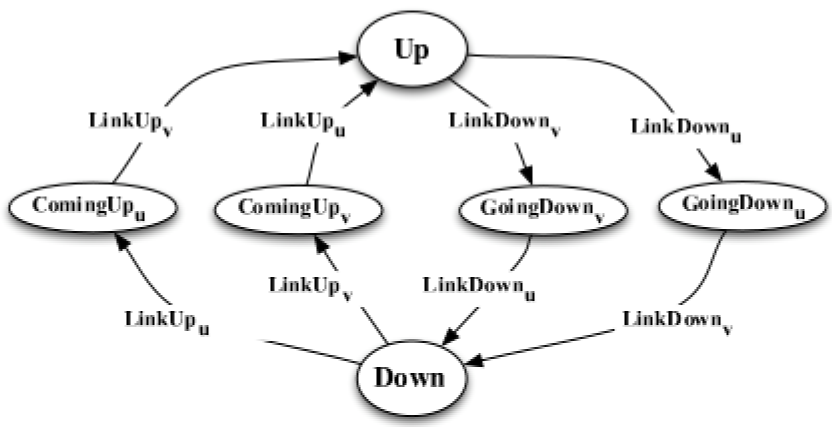

Figure 1: State diagram for status variable of $Link\{u, v\}$.

If the status is ComingUp u , then messages in transit from u to v are held in the queue until v has been notified that the link is Up. Once the link is Up, the event by which node u receives the message at the head of mqueue v,u is enabled to occur. An attempt by node v to send a message to node u is handled analogously.

Whenever a LinkDown u or LinkDown v event occurs, both message queues are emptied. Neither u nor v is alerted to which messages in transit have been lost due to the LinkDown.

In an initial state of the link, both message queues are empty and the status is either Up or Down.

1.3 Configurations and Executions

The notion of configuration is used to capture an instan- taneous snapshot of the state of the entire system. A configuration is a vector of node states, one for each node in P, and a vector of link states, one for each link in L . Assume that the undirected graph G = (V, E) defines the initial communication topology of the system, where V is a set of vertices corresponding to the set P of nodes, and E is a set of edges corresponding to the set of communication links that are up. In an initial configuration with respect to G, each node is in an initial state (as prescribed by the nodes algorithm), each link corresponding to an edge in E is in an initial state with its status equal to Up, and every other link has its status equal to Down. Define an execution as an infinite sequence C 0 , e 1 ,C 1 , e 2 ,C 2 , . . . of alternating configurations and events, starting with an initial configuration and, if finite, ending with a configuration, that satisfies the following safety conditions:

- C 0 is an initial configuration (w.r.t. some initial topology G).
- The preconditions for event are true in C_{i-1} for all $i \geq 1$.

- C i is the result of executing event e i on configuration C i1 , for all i 1 (only the node and link involved in an event change state, and they change according to their state machine transitions).

An execution also satisfies the following liveness condi- tions:

- If a link remains Up for infinitely long, then every message sent over the link is eventually delivered.
- For each link, if only a finite number of link events occur, then the link status after the last one is either Up or Down (not in between).

We also assign a positive real-valued global time gt to each event e i , i 1, such that gt(e i) < gt(e i+1) and, if the execution is infinite, the global times increase without bound. Each configuration inherits the global time of its preceding event, so gt(C i) = gt(e i) for i 1; we define gt(C 0) to be 0. We assume that the nodes do not have access to gt.

1.4 Problem Definition

Each node u in the system has a local variable lid_u to hold the identifier of the node currently considered by u to be one of the leaders of the connected component containing u, in such a way that the distance to this leader does not exceed a certain constant d (the remoteness constraint). The set of all the leaders including the supreme one forms a spanning tree as subgraph of the DAG established. In every execution that includes a finite number of topology changes, we require that the following eventually holds:

- Every connected component CC of the final topology contains a node l, the supreme leader, such that l is the only node which verifies $lid_l = l$.
- For each node u of each component CC, different from the supreme leader, a node v exists such as $lid_u = v$ and $d_{u,v} < D$ (D is the maximum remoteness towards a leader and the $d_{u,v}$ is the shortest distance between u and v)

In a more formal way, one can state the problem as follows:
In every execution that includes a finite number of topology changes, we require that the following eventually holds:

- For each node u of every connected component CC of the final topology: i selects (lid_u, d_i), $d_i \in N_i(CC)$ such that $(lid_i, d_i)_{i \in CC}$ is a spanning tree T (lid_i is the leader considered as such by i and N_i is path starting from i and whose vectors belongs to the set of those of the DAG.
- For each node u, different from the root of T, of every connected component CC of the final topology:
if $(k-1)D < depth_T(i) <= kD, k \in IN$, then $depth_T(lid_i) = (k-1)D$ ($depth_T(u)$ is the depth of u in T)

Our algorithm also ensures that eventually each link in the system has a direction imposed on it by virtue of the data stored at each endpoint such that each connected component CC is a leader-oriented spanning tree, i.e., every node has a directed path to its local leader respecting, among the other leaders, a certain hierarchy containing one supreme leader.