new node can proceed with MST construction only at the begining of a phase. This restriction may impose some delay before a new node can proceed with MST construction. We feel that this delay is worth the simplicity of the phase synchronizing scheme.

Next, consider how new edges can be added to the network. If these new edges came from at least one new node, then it will be introduced into a new computation phase at the same time by both the nodes at either end, as we have just seen. Let us now examine how an edge that once had an infinite cost now has a finite cost, i.e. it can be used for communication again. Note that the nodes on either end are not new and have been participating in computation phases, treating this edge as if it did not exist. Either or both the nodes will discover that the edge is available for communication and establish interprocess communication channels between them. The difficult part is introducing this edge with a new finite cost into the same computation phase for both nodes. If we assume that such edges come into existence only during the reinitialization process for both nodes, then either one of them could initiate reestablishment of the edge cost and the edge would be introduced in the new computation phase for both nodes at the same time. This assumption is necessary for the same reason that edges and nodes cannot go down during a computation phase, or that edge costs can not be changed during a computation phase. That is, the algorithm assumes a fixed topology during a computation phase and any change would cause the state information at various nodes to be inconsistent.

The assumption that no changes in topology can occur during a computation phase, and only during the reinitialization period is not unreasonable. The adaptive algorithm will be typically used in an environment where an MST is constructed, and used for a certain period of time and then reconstructed. Therefore each node will be in the reinitialization process for a longish period of time, deciding what the local topology should be like for the next phase. Nodes could be programmed to wait for a certain period of time when they have reinitialized themselves and found out that they are leaves. Hence, one can imagine that the nodes construct the MST, then spend some time reinitializing. Upon reinitialization the leaves of the MST could wait for some time before starting the new phase. Once they have started the phase, nodes that become leaves perform their usual functions. This does not require clocks in different nodes to be synchronized or even have the same period.

2.6.6 The Packet Radio Network Environment

We now describe how the algorithm can be used in the Packet Radio Network [Kahn75, Frank75] which uses centralized routing. Packets are forwarded from a source repeater along the branches of a tree to the station where they get routed either to a host connected to another network for which the station acts like a gateway, or to a user connected to a destination repeater. The tree along which packets are forwarded is rooted at the station and could be a minimum height tree. If a minimum spanning tree connecting the repeaters and station is

equally satisfactory, then this algorithm can be used to construct the MST when the repeaters are dropped from an airplane and the station is already on the ground. The repeaters and station all have the same algorithm executing in them. The algorithm adaptively recomputes the MST as more repeaters land and discover other repeaters. We assume that repeaters do not go down and edge costs do not become infinite (unless repeaters are in the reinitialization process!). We must, however, permit new edges to be introduced into the network at all times and not only during reinitialization. This can easily be done, as we shall see. Notice that the assumptions for the adaptive algorithm to work have not been violated in this real life application!

Assume that an edge (A,B) can go from infinite cost to a finite cost at any time. Either A or B, or both will discover that this edge is available for communication and establish interprocess communication channels between them. Assume now that A enters its reinitialization code because it gets a 'done' signal. A attempts to reestablish the cost of this edge. If B is also in its reinitialization code, then all is fine and the edge will enter the new computation phase, as we have seen in section 2.6.5. However, it is possible that B may alternatively be in one of the two following states:

(i) B may just have reinitialized itself and proceeded with the new phase (A and B could not communicate when B got its 'done' signal since edge (A,B) did not exist at that time).

(ii) B may not have yet got the 'done' signal.

A and B must synchronize their actions so that (A,B) has the same cost as seen by both of them for all phases. Let us see how this is achieved. A will be told by B that it is not in the reinitialization process and to wait for the response. A can not, however, wait indefinitely because B may have been in a state described in (i) above. A must treat this edge specially. A assumes that B is in state (ii) above and waits for a certain amount of time. If B responds in that time with establishment of the edge cost, all is again fine. If B does not respond, then A aborts this reestablishment and assumes that B was in state (i) and therefore treats the edge as though it had an infinite cost. A's assumption may have been wrong in that it just didn't wait long enough. In that case the situation and process B goes through upon getting its 'done' signal is symmetrical to what we just described. Hence, if timings are not right, then it is likely that A and B will not introduce this new edge into the network for a number of phases. We believe that nodes will be in the reinitialization process for a longish period of time, and that there are stochastic delays and so this synchronization will eventually come about. This synchronization mechanism is very much in the same spirit as the one used by the Internet Transmission Control Program when it sets up an interprocess communication channel across a very unreliable subnet [Cerf74a, Tomlinson74, Dalal74, Dalal75, Sunshine75].

As repeaters land they will discover the world around them. They do not wait for all repeaters in their neighborhood to land since they

do not know how many there will be. A repeater decides that it has enough neighbors and then considers itself initialized and proceeds to construct the MST. It is possible that a small MST will be constructed in the first phase, and this will become larger as new nodes are added in subsequent phases. It is also possible that a number of small MSTs will be constructed, but as more nodes land or the existence of new edges are discovered the small MSTs will connect themselves to one another producing a final MST.

2.6.7 Analysis of the Algorithm

The adaptive algorithm is relatively simple once a new computation phase has been properly initiated. In terms of the abstract parallel algorithm, leaf nodes decide to connect the new fragment they have information about to another fragment by the minimum cost edge. Since all N nodes eventually become leaves and there are only N-1 old-branches, one and only one node determines that the computation phase has terminated. This node informs the others by broadcasting a 'done' signal, along the branches of the MST just constructed.

The reinitialization process that a node undergoes upon realizing that the current computation phase is over is very important. The properties of the protocol by which edge costs are established have been described in sections 2.6.3, 2.6.4, and 2.6.5. The algorithm assumes that certain changes in topology, i.e. edges or nodes going down, only occur during reinitialization, in order to keep the topology from changing during a computation phase, and to guarantee that a computation

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.