Proc. 9th ACM Symp. on Principles of Distributed Computing, Quebec City, August 1990

Distributed control for PARIS
Extended Abstract

Baruch Awerbuch* Israel Cidon† Inder Gopal†‡ Marc Kaplan†§ Shay Kutten†

Abstract

We describe the control protocols of the PARIS experimental network. This high bandwidth network for integrated communication (data, voice, video) is currently operational as a laboratory prototype. It will also be deployed within the AURORA Testbed that is part of the NSF/DARPA Gigabit Networking program.

The high bandwidth dictates the need of specialized hardware to support faster packet handling and control protocols. A new network control architecture is presented which exploits the specialized hardware in order to support the expected real time needs of future traffic. In particular, since control information can be distributed quickly, decisions can be made based upon more complete and accurate information. In some respects, this has the effect of having the benefits of centralized control (e.g. easier bandwidth resource allocation to connections), while retaining the fault-tolerance and scalability of a distributed architecture.

*Dept. of Mathematics and Lab. for Computer Science M.I.T., Cambridge, MA 02139. The work of this author was done while visiting IBM T.J. Watson Research Center.

†IBM T.J. Watson Research Center, Yorktown Heights, NY 10598.

‡IBM T.J. Watson Research Center, Yorktown Heights

§IBM T.J. Watson Research Center, Yorktown Heights

1 Introduction

Packet switching networks have changed considerably in recent years. One factor has been the dramatic increase in the capacity of the communication links. The advent of fiber optic media has pushed the transmission speed of communication links to more than a Gigabit/sec, representing an increase of several orders of magnitude over typical links in most packet switching networks ([KMS87]) that are still in use today. Increases in link speeds have not been matched by proportionate increases in the processing speeds of communication nodes. Another factor is the changed nature of traffic carried by these networks. As opposed to solely data networks, or solely voice networks, it is now accepted that packet switching networks (or variants of packet switching networks like ATM ([Kil87])) will form the basis for multimedia high speed networks that will carry voice, data and video through a common set of nodes and links.

The disparity between communication and processing speeds suggests that processing may become the main bottleneck in future networks. To avoid this possibility, these networks will be built with high speed switching hardware to off-load the routine packet handling and routing functions from the processor ([CGK88]). In addition, the real time traffic (e.g. voice) requires that the route selection function be capable of guaranteeing the availability of bandwidth on the links along the chosen path for a particular traffic stream. Otherwise, congestion will cause excessive loss of information. Both of these factors have a significant impact on the design of the protocols and the control procedures for the network. In this paper we explore this impact, specifically focusing on the lessons that we have learned during the design and implementation of the PARIS network ([CG88]). We believe that most of our conclusions are general and can be applied to any high speed packet network including ATM based systems.

The early stage of PARIS was described in [CG88]. That work included some initial ideas regarding distributed control. In the current paper we describe for the first time the distributed control functions, and they way they fit into the complete network. We also introduce a new multicast feature that is to be implemented in hardware, and exploit it in performing fast hardware broadcast.

The control procedures of the PARIS network facilitate virtual circuit routing. Thus, we have the notion of a "connection" or "call" being established between a source and a destination. For each call, all the traffic of that call traverses the same path through the network. The control process is as follows. Requests for "calls" arrive at nodes asynchronously. Each call has associated with it some parameters such as average packet rate, burstiness, tolerance for packet loss, etc. Calls are either denied access to the network ("blocked") or accepted into the network. If accepted, a call is provided with a route that has adequate capacity to handle the request. The mechanisms used by each node to perform these functions are referred to as the control procedures of the network.

Traditional data networks typically employ distributed control but do not guarantee availability of bandwidth to calls. Control procedures in common carrier networks (circuit switched networks) deal with capacity allocation but are typically more centralized and rely on the availability of significant computing power. A key contribution of PARIS is showing that by employing hardware speed-ups in the control flow it is possible to provide performance guarantees while preserving the fault tolerance and growth capabilities of a distributed control architecture.

The work presented in this paper is more than a "paper study". Considerable prototype implementation has been done and much more is planned. A prototype PARIS network, operating at switching speeds of over 1 Gbps, has been built and tested within a laboratory environment. In the near future, more realistic deployments are planned. For example, a PARIS network will be installed in the AURORA testbed. Many of the mechanisms described in this paper will be implemented and experimentally validated as part of that project. The AURORA project is part of the NSF/DARPA Gigabit networking program, under the auspices of the Corporation for National Research Initiatives. It will involve the construction of a Gigabit network that will link together four research centers in the Northeastern United States (MIT, University of Pennsylvania, IBM and Bellcore). It is likely that the results of this project will provide considerable experience and understanding of how distributed control algorithms will operate in future networks.

2 Network model

Each PARIS node consists of two components, a fast hardware switching component (the Switching Subsystem, SS) and a slower controller (Network Control Unit, NCU) (see Figure 1). The SS performs the simple packet routing functions while the NCU performs the more complex functions. Bidirectional transmission links are attached directly to the SS. The NCU is also attached to the SS by a bidirectional link. We assume that each link has a finite, non-empty, set of ID's. The hardware permits each link's ID sets to be configured dynamically under software control. In this paper we will assume that the various ID sets are defined in order to perform the following functions.

1) ANR routing: This requires every link to have

an ID that is unique within its SS. If a certain node (where no ambiguity exists, we shall say "node" instead of "the NCU in a node") wishes to send a packet to certain destination node and if it has knowledge of a path to that destination node, it can send the message by prefixing the data with a string that is composed of the concatenation of all the link ID's along the computed path. (See figure 2).

2) Selective copy: Assume that for each link attachment (excepting the NCU's attachment) we define a "copy ID" that is identical to the (primary) link ID except for its most significant bit. For each link, both the copy and the primary link IDs are configured as members of the ID set. By also assigning all of the copy ID's to the ID set of the NCU's attachment, it is possible to achieve a selective copy function - A packet may be copied by several preselected nodes along a path by substituting the copy ID for the normal ID with the ANR for these preselected nodes. (See figure 3).

3) Multicast: If more than one link recognizes the same ID, it is possible to perform a multicast within the node. This feature is exploited in the tree broadcast procedure used for topology update. (See figure 4).

As previously mentioned, the basic unit of traffic is a "call" or a "connection". ¿From the viewpoint of the control procedures, a call is defined to be a stream of packets from some external source with a specified average rate, variance, service requirement, etc. We enforce that each source restricts itself to the specified parameters through an input rate regulation scheme [CG88]. The "leaky bucket" scheme proposed in ([Tur86]) and the credit manager scheme used in SMDS ([SLCG89]) are other examples of a input rate regulation schemes.

The basic operation of the PARIS input regulation scheme is simple. Input packets first (attempt to) enter a queue, Q1. If the queue Q1 is full the packet is discarded. In order for the packet at the head of the line to enter the network, it must obtain a token from a token-pool. Tokens are generated into this token-pool at fixed time intervals that correspond to the specified average rate of the connection. If a pre-defined maximum number of tokens (say, M) have been collected in the token-pool then the token generation process is shut-off. The scheme guarantees that the long term average rate does not exceed the pre-specified rate of the connection. However, over shorter periods, it permits bursts at a much higher rate. The choice of M determines the burstiness of the transmission and permits us to control the variance of the arrival process into the network. In ([SLCG89], the precise effect of these parameters on the characteristics of the input process is analyzed. Further development of this scheme was already conducted [BCS90].

3 Control cycle

The PARIS approach to connection control is a decentralized one. This design choice is motivated by the fact that PARIS is aimed at private networks rather than public carrier networks. For fault tolerance and performance reasons, it is well accepted that for such networks decentralized control is preferable to reliance upon one or more central controller(s) [BGJ+85]. Thus, in the PARIS system, every backbone node participates in a set of distributed algorithms which collectively comprise the connection control of the system.

While distributed control mechanisms are commonly used in most of today's data networks [BGJ+85, MRR80], they do not deal with traffic that requires service guarantees. As mentioned in the previous section, an input throttle is used to regulate the traffic rate and it is assumed that all traffic that passes through the throttle is guaranteed a certain level of service. Thus, before admitting a call into the network, some guarantee must be provided that the communication capacity required by the call is available. If not, the call must be denied access into the network or "blocked".

The basic design choice in PARIS is a distributed route selection mechanism using a routing topology database similar to the one in ARPANET [MRR80]. Basically, each node maintains a routing topology database with link weights reflecting the traffic over each link. When link weights change substantially updates flow to every node using a broadcast algorithm. At the call setup time, the source node obtains the parameters associated with the new call. These parameters define the type of call, the destination, and the parameters of the input throttle that is associated with this connection (specifying either directly or indirectly the average capacity of the connection and the level of burstiness). Typically, these traffic parameters are are based on traffic type (e.g. a voice call requires a steady 64 Kbps) and may be changed dynamically during the operation of the connection. The source node then computes a path based on its local routing topology database generates the ANR field from source to destination and back. The source node uses the information in the local topology database to ensure that the chosen route is capable of carrying the traffic and providing the level of service required by the traffic type. The computed information is then sent to the adaptor that actually interfaces with the source of traffic. The call setup procedure is then initiated. As part of the procedure, an end-to-end call setup packet flows over the path and is copied by the intermediate nodes along the path. Based on the bandwidth information in the call setup packet, each of these nodes updates its database of the bandwidth usage on its link attachments. This updated information may change the link weights and trigger an update broadcast.

If no suitable path can be found between source and destination, the call will be blocked. The scheme provides control of the path at the source and obtains relatively efficient paths. However, because the information about remote link utilization takes a non zero time to propagate through the network, there is a possibility of some unnecessary blocking caused by temporarily inaccurate information in the routing topology database. In PARIS, we employ an efficient way for performing the topology update - we use a fast tree broadcast function. This new feature also reduces the message processing overhead within each node. Using the speed of the network hardware, this fast broadcast reduces the problem of transient inconsistencies in the topology databases.

The process of connection control can be captured in the form of a "control cycle" shown in figure 5. The cycle represents the flow of information in the system. The cycle starts with a request for a new connection. This request contains the call parameters. The information used to compute a route for the call comes from the local topology database. This information is obtained through the topology broadcast/update algorithm. The trigger for the topology broadcast comes from the local link weight computation of each node. These weights are computed from the knowledge of the call parameters for each of the calls that traverse the links, knowledge that is gained during the call setup process. The initial source of the parameters is, the connection request. This closes the cycle.

Note that two components of the cycle involve interactions among several nodes. These are the call setup process and the topology broadcast. We use the fast copy and the fast broadcast capabilities of the Switching Subsystem hardware to speed up the operation of these two critical components of the control cycle. In the rest of this paper, we discuss briefly the various components of the cycle. They are presented in this paper for the first time.

4 Route computation

Recall that since the full topology is known at each network (NCU) node this is essentially a local operation. This procedure also determines whether or not a given call is permitted access to the network. While the scheme is basically a collection of heuristics, the underlying "optimality" criterion or long-term objective is to maximize network throughput subject to the maximum loss probability allowed by each packet. (Packet loss increases with throughput.) Unlike the case in traditional networks, minimizing delay is not an objective, since in a fast network packet sent and not lost will arrive fast.

We use several "rules of thumb" to guide us in the development of the route computation scheme. For example, the route computation method should attempt to find a path with the minimum number of hops as this minimizes the use of network capacity. Also, calls with excessive capacity demands should be denied access to the network. We define "excessive" by comparing the number of hops in the current route with the "minimum hop" route. This criterion is particularly important under high load conditions and for calls with large holding times.

The resulting scheme is sketched as follows. Based on the characteristics of the call the first step is to identify the set of links that can accommodate the call. Given the characteristics of the call, and link weights in the topology database, it is possible to compute the expected packet loss (the primary parameter in determining acceptability of a link). Among the subset of acceptable links, a minimum hop path is then chosen. If the length of the chosen path is within a permitted multiple ("stretch factor") of the minimum possible path length the call is admitted into the network. (The "stretch factor" is determined by the current load conditions and input call parameters). Otherwise, the call is blocked. (This may require some calls to be blocked even though resources are available, in anticipation of future calls which are expected to make better use of the resources.) Further investigation of this ideas is being conducted [ACG].

5 Information Update

5.1 Overview

Each local node is responsible for determining the bandwidth utilization of its adjacent links, for determining when to inform remote nodes of changes in utilization, and for distributing this information to remote nodes. Distributing this information is called a *utilization broadcast/update*. A similar task is the *topology update* where the information about the active and inactive links is distributed.

For both updates it is possible to use a conventional flooding based mechanism as in ARPANET [MRR80]. However, the ARPANET algorithm has some deficiencies that make it sub-optimal for this purpose. Firstly, it delivers a copy of every message over every link. This means that each node has to do the extra work of handling redundant packets. (It is enough that each node receives only one copy of each message.) Secondly, this algorithm is hard to implement in fast hardware. (Remembering which message has been received before is a task that hardware switches currently cannot perform fast and cheaply. Thus the relatively slow NCU in each node must decide whether to forward a received message, or to discard it as a copy.) This results in making the algorithm too slow to operate in a rapidly changing traffic environments.

The selective copy mechanism can be used to perform a multicast or broadcast (e.g. through a path that traverses a Depth First Search. See e.g. [Eve79].) This, however, has drawbacks in that it results in very long paths (and consequently long messages' headers) and it requires that the sender must know a route that reaches all the recipients of the message. In the task of *topology/utilization update* every node may be a recipient. Any node who notices a topology change may be the broadcast source. Clearly we cannot assume that the route to all nodes is known to every node, since otherwise no topology update would have been needed. Another methods to broadcast using ANR only have been suggested in [CGK88]. The method used here is faster, since it uses a new hardware feature.

The *multicast* message mode was introduced mainly for this purpose. Recall from a previous section that a link adaptor may have multiple ID's (labels) and that these ID's can be changed dynamically by the local NCU. Suppose that some link adaptors, in various nodes, have (among other labels) the label T. Assume further that the collection of the T labeled links forms a tree. This is used for fast hardware as follows. A $Tree$ type message, using label T, arriving at a switch over a link labeled T is forwarded over all other links labeled T of that switch. Note that this broadcast will reach every node on the tree, and will terminate, with no need for sequence numbers.

Topology updates are triggered whenever a node senses the failure or recovery of an adjacent link. Utilization updates are triggered whenever the node senses that the utilization of an adjacent link has changed substantially from the time of the previous update. Utilization updates are also sent periodically (as described below) to guarantee reliability.

The $Tree$ type messages as defined above have no built-in error recovery mechanism. There is some finite (very small) probability that a message sent on the tree will not arrive at some of its destinations. In both the topology and the utilization update tasks we make use of a "backup" periodic broadcast of utilization updates to achieve reliability. The periodic approach is suitable for such tasks because it is important to receive only the most recent link information (previous updates becomes obsolete once a new one is received). (Note that a link utilization message is also implicitly a link topology message. A link that is utilized must be up). The periodic broadcast is achieved by having each node maintain a time-out period and performing a "periodic" broadcast if no event driven utilization broadcast has occurred within this time period. Note that we expect utilization updates to be very frequent and do not expect this periodic mechanism to be triggered very often.

As we would like to use a hardware broadcast tree for the topology maintenance protocol, we need a mechanism to enable the nodes to correctly and consistently label their adjacent links as tree links or non-tree links. Since every node maintains a network topology, it seems that it could have computed a tree according to some procedure that is consistent among all nodes (e.g. a minimum spanning tree) and thereby know how to label its adjacent links. This simple approach does not work in a dynamically changing network as it may result in transient loops in the tree labeled links. This will cause looping of messages through the hardware and excessive traffic in the network. Thus, we introduce a tree maintenance procedure that uses network topology but imposes careful coordination between the nodes in order to ensure that transient loops do not occur.

Our complete protocol is composed of two modules. The topology update module (and the utilization update as well) broadcasts over the tree computed by the tree maintenance protocol. On the other hand the tree maintenance module uses the topology knowledge in order to maintain the tree (see Figure 6). That is, when the tree is disconnected the replicated topology database is used to locate the edges that will reconnect the tree. The topology update and the tree maintenance algorithms are described in Appendix A.

Let us comment that a tree maintenance protocol appears in [ACK88]. This protocol cannot make use of a hardware broadcast, since it assumes reliable delivery of messages (that our basic broadcast does not provide). Also, the protocol of [ACK88] is more complex than required here, since the use of sequence numbers of messages is not allowed there. (It leads to theoretically unbounded message length. For practical purposes, though, 64 bits numbering is more than enough.) In addition to simplifying the protocol, the use of numbering also enables a cheap dissemination of the topology. This availability of the topology information further simplifies the task of tree maintenance module. (The protocol here is also better in communication in a (practically important) constant factor.)

The topology knowledge also enables us to adjust the tree easily to be a minimum spanning tree (rather than any spanning tree). It also helps us to achieve a stabilization of the tree even in the presence of some less reliable links (their weights will be increased).

5.2 Load Balancing

In a high speed environment where links are very reliable and of very high bandwidth, we expect utilization information to change at a rate which is several orders of magnitude faster than the link topology information. (Seconds vs. hours or days.) Therefore, the utilization update messages dominate the total update traffic and are the main concern of this section.

A key issue to ensure avoidance of congestion in the process of utilization update is that of a load-balancing. If every node is permitted to send utilization updates at any time we may run into potential problems. It is possible that even if the processing rate at a node is adequate on average to cope with the total rate of updates, there might be periods in which the number of concurrent updates exceeds the processing speed causing the update message queue to become congested. Adding to this problem is the fact that utilization updates at different nodes might be correlated. The reason for that is the introduction (setup) of high-bandwidth calls through a long path of nodes. Since the call setup is almost instantaneous at all these intermediate nodes they may issue a utilization update at almost the same time. Another scenario is a failure of a high bandwidth link which cause a rerouting of a large number of calls all within a short period. This means that even though our broadcast media is not a collision type we would prefer to spread updates over time in order to guarantee load-balancing.

We employ a scheduling mechanism (BRAM) that is usually used for scheduling transmissions in a shared media network (radio or coax), see [CFL79]. In such a scheme the scheduling is done by ordering the transmission of nodes in a round robin fashion. Nodes that have nothing to transmit are skipped dynamically by detecting the absence of their transmission.

The implementation of the BRAM algorithm in the network is straightforward. The ordering of the transmission can be done locally by each node using its replica of the topology database and the topology of the broadcast tree. The node can also estimate the propagation delay through each link and in particular the tree links. Inconsistencies between the replicated topology databases can be ignored since the network can tolerate "collisions". The BRAM algorithm works better if the sum of the propagation delays between consecutive nodes of the round-robin schedule is minimized. If the broadcast were done on a general network this would pose an NP complete problem. However, since the broadcast is performed over the hardware based tree this problem is solved by a simple Depth-First-Search procedure. Since our broadcast mechanism is collision free and the delays are only estimated, the BRAM algorithm is only approximated.

6 Call Setup, Maintenance and Termination

Traditionally, in virtual circuit based networks, (i.e. TYMNET, X.25 etc.) call setup and termination procedures are used for two different tasks. First, the intermediate nodes must update their label swapping table in order to activate the intermediate switching operation for a specific call. Second, the two end-points must establish an end-to-end connection and exchange session parameters (i.e. window size, packet size etc.).

In the PARIS system, since we use ANR routing, there is no need for any table update to allow the physical communication. The end-points are able to communicate once the routes have been computed. However, for bandwidth management reasons we are using the call setup/takedown procedures as the mechanism to inform the intermediate nodes about the amount of bandwidth that is allocated to the new calls. (Recall (Section 5) that the nodes track the amount of reserved capacity for each of their local links and broadcast a utilization update if some significant change has been identified.)

Another task of the setup procedure is to reconfirm the availability of the reserved bandwidth for the new call. This task is sometimes necessary because of the potential latency in the operation of the bandwidth control cycle. Calls which are concurrently routed from different sources may allocate capacity from some link without being aware of each other. Typically, this will cause no harm if the call bandwidths are small compared to the residual available capacity. However, for congested links or high bandwidth calls (such as high quality video calls) this might cause an over-utilization and hence an excessive packet loss.

We also introduce the concept of a call maintenance procedure in order to satisfy the following requirements:

1. To track in real time the bandwidth being reserved and released in order to pass significant load changes to the topology/utilization broadcast mechanism.
2. To notify the end points of a call about failures along the call's path that require a call drop or a switch-over to an alternate path.
3. To release the reserved capacity of explicitly terminated calls (with explicit take-down messages).
4. To eventually release the reserved capacity of implicitly terminated calls. The absence of an explicit take-down message can be caused by:

   Failure of the end-points before the normal termination of the call.

   Link/node failures that isolate the intermediate node from the end-points of the call.

In addition to the above we use the call maintenance procedure to enhance the fault-tolerant operation of the network. Since the SS is a stand alone hardware module, independent of the NCU, the failure of the NCU does not necessarily impact the flow of steady-state traffic of existing calls. (This failure, however, will prevent the setup of new calls through this node). Thus, a recovering NCU (or a to a backup NCU) may not "know" the reserved bandwidth and the actual capacity used in its links. We introduce a mechanism by which such a processor can first regain the reservation information and then rejoin the call setup process.

6.1 Setup/Takedown

The call setup procedure is composed of two complimentary phases. They are described in details in [CGS90]. In the first phase the source of the call notifies the destination and the intermediate nodes along the path of the new call and its characteristics. This phase is accomplished by the source sending a direct message to the destination which is also copied by the intermediate nodes (using the selective copy mechanism).

The second phase includes a call confirmation process in which a confirmation message is transferred through the intermediate nodes back to the source. Each node checks whether the reserved capacity is indeed available. Otherwise it will convert the confirmation message into an abort message. The confirmation phase is optional in the sense that in most cases the source does not wait for the confirmation message before end-to-end communication is enabled. However, the reception of an abort message will cause the session to be immediately aborted. (The confirmation process can be accelerated by having the nodes on the way send confirmations in parallel using ANR. Alternatively, some nodes along the way may accumulate some downstream confirmations, and sending a single consolidated confirmation [CGK90].)

The call termination is very similar to the call setup without the confirmation phase. The reserved capacity is released. Since the call might be terminated by external events such as failures along the path we must have other ways to terminate the call and to release the reserved capacity in such events. These mechanisms will be discussed in the following.

6.2 Call Maintenance

Our call maintenance procedure is to have each source periodically send refresh messages which include the call parameters. These messages are immediately acknowledged by similar messages from the destination. These acknowledgment messages are also copied by intermediate nodes. The basic assumption of this scheme is that in the absence of failures, the packet loss probability for control messages is very small and thus the probability that some fixed small number of such consecutive messages (K - in the range of 2-5) will be lost is practically negligible [CGGS88].

The periodic message exchange is used in several ways. First, it serves as a path integrity check for the end-points of the session. The absence of a refresh message indicates to the end-points a failure along the path or the failure of the other end-point. Second, these messages allow the intermediate nodes to track the existence of calls and the amount of bandwidth reserved for these calls. (This will be further explained later on). This requires the copy of the refresh messages by the NCU. The time requirements for this task are less strict than the first one. Therefore only a known subset of the refresh messages should be marked as messages to be copied. Third, this periodic transmission of the call parameters allows nodes to refresh their reservation knowledge or automatically recover it after a NCU crash just by processing the copied refresh messages for some period of time.

6.2.1 Tracking bandwidth reservation

There are two basic approaches to the reservation refresh procedure: (1) explicit; and (2) implicit. In the explicit refresh the NCU maintains a call table in which each call ID has an explicit entry that describes the amount of capacity reserved. A timeout period is maintained for each entry. After the reception of a refresh message for a specific call ID the timer for that entry is reset. If after some predetermined number of refresh periods (considerably larger than the above K) no refresh is received, the call is considered terminated and its entry is removed from the table. We assume that a similar but shorter timeout period is used by the end-points so they will drop the call before the intermediate nodes. This ensures that actual transmission ceases before the call capacity is released.

The drawback of the explicit refresh is that a large amount of memory and processing is required. A typical high speed link (say SONET STS24 which is approximately 1.2GbPS) may carry over 20,000 64KBPS phone calls. The duration of a voice call is usually around 200 seconds which leads to a refresh time for the reserved capacity of about 1-10 seconds. This results in 2,000-20,000 such operations (which include finding the entry in the table, resetting the timeout flag, and other overheads) per second per link and a table size of 20,000 entries per link.

For the majority of calls we employ an alternative implicit approach which is less exact but also less computationally expensive. The idea is that over some sufficiently large "window" of time (which will depend on the maximal difference in the delay between consecutive packets) the number of refresh packets that will be received is fairly constant. (In our example the window can be set to 10 refresh periods. Using the law of large numbers, for 20,000 calls the number of refresh messages received in a window will be very close to $10 \cdot 20{,}000$ with very high probability). Here, we do not have to maintain an individual table entry per call but simply need to keep the sums of the reserved capacity for the last window. A weighted sum of these short term estimations serves as the node estimate for the link utilization.

A potential problem exists since some calls may be of a very high capacity and therefore the total implicit sum will depend much more on their refresh messages than on the messages of the other calls. This causes the law of large numbers not to hold for the sum (it will still hold for the total number of refresh messages). In PARIS, we make such calls use the explicit refresh procedure and these refresh messages are not taken into account for the implicit summation. A typical rule is that if a call requires more than x% of the total link capacity (say 1%) than this call will be explicitly maintained in the reservation table and will not be part of the implicit summation. Thus, we have a strict upper bound on the number of calls we maintain explicitly (say 100). The resulting hybrid scheme is illustrated in Figure 7. A different approach would be to break the large capacity calls into smaller pieces and instead of sending a single message per refresh period to send a multiplicity of refresh messages each carrying a fraction of the total call capacity.

A further way of reducing the computational burden of the NCU is to introduce additional hardware associated with each link. Since the processing is trivial, the refresh messages can be processed on-the-fly by some special purpose module which will be part of the link adaptor hardware. Only sums will be reported to the NCU.

A APPENDIX: Topology Update Algorithm

In Subsection A.1 we describe the tree maintenance module assuming that the topology maintenance module ensures that nodes on a tree know eventually the topology of their tree, and neighboring edges. In Subection A.2 we explain how this assumption is realized.

We use the graph representation of the network, and refer to the actions of the NCU as being actions of nodes.

A.1 Tree Maintenance

For any link (v, u) the only node who can put it into the tree is its endpoint node v. (Node u does similarly for edge (u, v), so each edge has two entries in the tree description.) This is done by (1) Updating the local topology map to show that this is a tree link; (2) Putting the tree's label on the adaptor, in the switch (SS), so that the switch will know to forward update messages over it; and (3) notifying the other nodes so that they too can update the status of the link (to be "tree") in their local topology map. Tasks (2) and (3) are performed by generating a topology change (i.e. the edge changes from non-tree status to tree status) which will be handled by the topology update protocol. A node w that gets a notification from the topology update protocol about an edge (u, v) becoming a tree edge updates its topology map. Similarly a topology update may be either the failure or the recovery of a link, or the fact that it has stopped being a tree link.

Note that when the tree is temporarily disconnected (e.g. due to a failure) it is actually a forest of node-disjoint trees, rather than a single tree. The protocol strives to make this forest into one tree that spans the whole (connected component of the) network.

Each node also remembers which of its tree edges leads to its Parent. The protocol keeps the values of these Parent variables (in different nodes) consistent in the sense that if node v is the Parent of node u, then u is not the Parent of v. (The one exception is the time that a message from u is on its way to v to transfer the parenthood from u to v). Thus each tree has a single node with no Parent. This node (whose Parent = nil) is the tree *root*. It coordinates the effort of the tree to merge with another tree. These mergings are repeated whenever possible. Thus, if the network is stable for a reasonable amount of time all the trees in a connected component of it merge into one tree. (It has been estimated that in very large networks a topological change will take place about every one hour.)

Using the Forest database, a root r knows which nodes belong to the same tree as itself. Using also the topology map root r can find whether there is an edge connecting a node in this root's tree to a node that does not belong to that tree.

We assume that each edge has a unique weight, known to both its endpoints. This can be achieved by using concatenation of the names of its endpoint nodes as the least significant part of the edge's name. The description of the more significant parts is deferred to Subsubsection A.1.1.

Let (k, j) be the edge with the lowest weight among those edges connecting root r's tree to a node not in this tree. (Call it the minimum *outgoing* edge.) If k is not r then the "rootship" is transferred to k by sending messages and changing the values of the Parent variables in the nodes on the way to k. Performing this transfer by a direct ANR message from r to k is somewhat tricky. Consider the case that k receives the message and then a disconnection of a tree link (v, w) occurs on the tree path from r to k. If node w has not received the message then it will consider itself a leader of the new tree. Also, any node on the path that has not received a copy of these message will not change its Parent variable to point in the direction of k. Thus k must first verify that the message has been received by all the nodes on the path (except for nodes that became disconnected from k due to a tree link disconnection). This is achieved by treating the rootship transfer message similar to a call setup (See Section 6)). Here we prefer the fast check version of direct ANRs. (If the path is very long then the version of [CGK90] can be used.)

Note that during this transfer the minimum outgoing edge may change (by the failure of edge (k, j), or by the recovery of another edge). This is detected by the new root that transfers the rootship to the endpoint node of the new minimum edge.

When the root is the endpoint of the minimum outgoing edge, it negotiates merging with the tree on the other endpoint of the edge. For the merging it is required (similar to [GHS83]) that both endpoints will be the roots of their trees, and that both will agree on the merging. (This is introduced in order to prevent entering cycles into the tree.) The root with the lower identity suggests the merging, and waits until the other endpoint agrees.

Similar to [ACK88] this suggestion is canceled in the case that the edge fails. It may also be canceled when a lower weight edge recovers. In this case the suggesting root must first ask the other root whether the suggestion has already been accepted. If the other root has already agreed to the suggestion, then it is not canceled. Otherwise the suggestion is withdrawn, and the suggesting root is free to suggest a merge over another edge.)

When both sides agree to the merge the topology update protocol is invoked (to exchange topology information between the nodes, and to notify about the new edge). Finally the trees are connected by having both sides put the edge in the forest as described above. The side with the higher identity (say j) remains a root, while the other, k, sets its Parent Variable to point at j.

A.1.1 Links weights

Finally let us say a few words on the weights of the links. A field in the weight that is more significant than the nodes names is the links speed (faster links are assigned lower weights) in order to prefer fast links in the tree construction. Consider a node u that is an endpoint of a "heavy" link (u, v) (e.g. a T1 [KMS87] or slower link) who learns (from the topology update) about the existence of a lower weight link that should replace link (u, v) in the tree. (We expect that the tree links will usually be SONET OC3 [BC89] and above.) Node u removes link (u, v) from the tree. This is a topology change. Thus the tree root learns about it (from the topology update protocol) and moves to mend the tree. Note that the better link will be put in the tree this time. In order to prevent excessive changes to the tree node u removes link (u, v) from the tree only if the difference between the weights of link (u, v) and the new link is above some predefined value.

Note also that a link that goes up and down frequently can activate the protocol again and again to connect the tree. If this link has a small weight then this can make the protocol so busy so that it will not respond to the failure of other links. (This is a case of *starvation*). This is an unlikely case, since the detection of a link failure is rather slow [HK89]. Still it is prevented by putting the reliability of the link as the most significant field in its weight. Each failure of a link increases its weight. Note that this weight may now not be consistent in the endpoints of the link as long as the link is disconnected. However, for our protocol only the weight of links that are *up* (not disconnected) matters. (When the link is brought up, its two endpoints, and hence their trees, agree on the weight. This is considered a topology change, and hence triggers broadcast).

A.2 Topology update

First let us assume that no messages are lost by the fast broadcast.

Each node v has a single counter Seq-No (initially zero) (the *sequence number*). Whenever v notices a topological change in one of v's edges (v, w) the value of Seq-No is incremented. The topology update item generated includes the new description of the edge (up, down, tree, ...), and the new value of Seq-No. This item is broadcast over the tree that includes v.

Recall (Subsection A.1) that when two trees merge their roots invoke the topology update protocol, to exchange topology information. To reduce the traffic, they first exchange their Seq-No-Vectors. This vector in a node v contains, for every other node k, the sequence number of the last topology update received from v. (Its value is updated with the reception of each update generated by v.) Next each side sends the other the utilization information the other missed. This is detected as follows. Assume side v (root v) has value x in its Seq-No-Vector(k) and in side j the value of Seq-No-Vector(k) is y. Assume further that x is larger than y. Then node v sends node j all the utilization updates generated in node k with sequence numbers greater than y (up to x). Node j then broadcasts this information on its tree.

The hardware broadcast will be much faster than the transfer of the rootship. (The hardware switch obeys the FIFO discipline.) Assume that a node receives a hardware broadcast and then is disconnected from a certain tree. By the time it is reconnected, the broadcast is terminated and the node will not get the same message again. The Seq-No exchange also costs at most one item exchange per each topology change.

Let us now consider the case that messages may be lost by the hardware broadcast. There is a backup mechanism that is "piggybacked" on the utilization updates (Section 5.2). They also carry topological information. It is highly unlikely that a node will not have updates to send about its link in a long time. (See Section 5.2.) If this does happen, it will send an update nevertheless, thus eventually every connected node hears about an active link. Assume, however, that a node has not heard about the disconnection of a link. Some algorithm (e.g. the tree maintenance, or the call setup) may try to use the link. It will then detect that the link is not there, and invoke a topology update.

Up to now we have ignored the case that a node fails, and its Seq-No is erased. This is an unlikely case, since we use a non-volatile memory for the sequence number. For the case that it does happen we have added a random number 64 bit field to the node's name. When a node recovers it chooses the value of this field at random. This will differ from the previous name with an overwhelming probability. Thus the new topology updates sent by this node will not be discarded by other nodes, even though they have small sequence numbers. A name that is no longer used will disappear, due to a slow time driven backup update and garbage collection protocol. This backup is used anyhow to let a node "forget" about another node from which it becomes disconnected forever. Similarly, a node that has not heard a utilization update of some edge for a long time, assumes that this edge is disconnected. (This backup protocol will actually recover also from the extremely unlikely case that the chosen node name is the same as the previous.)

References

[ACG] H. Ahmadi, J.S.-C. Chen, and R. Guerin. Dynamic routing and call control in high-speed integrated networks. Unpublished manuscript.

[ACK88] Baruch Awerbuch, Israel Cidon, and Shay Kutten. Communication-optimal maintenance of dynamic trees. Unpublished manuscript, September 1988.

[BC89] R. Ballart and Yau-Chau Ching. Sonet: Now it's the standard optical network. *IEEE Communications Magazine*, 27(3):8–15, March 1989.

[BCS90] K. Bala, I. Cidon, and K. Sohraby. Congestion control for high speed packet switched networks. In *INFOCOM'90*, June 1990.

[BGJ+85] A. E. Baratz, J. P. Gray, P. E. Green Jr., J. M. Jaffe, and D. P. Pozefsky. Sna networks of small systems. *IEEE Journal on Selected Areas in Com.*, SAC-3(3):416–426, May 1985.

[CFL79] I. Chlamtac, W. Franta, and K.D. Levin. Bram: The broadcast recognizing access method. *IEEE Trans. Commun.*, COM-27:1183–1190, 1979.

[CG88] I. Cidon and I. S. Gopal. Paris: An approach to integrated high-speed private networks. *International Journal of Digital & Analog Cabled Systems*, 1(2):77–86, April-June 1988.

[CGGS88] I. Cidon, I. Gopal, G. Grover, and M. Sidi. Real-time packet switching: A performance analysis. *IEEE Journal on Selected Areas in Communications*, 6(9):1576–1586, December 1988.

[CGK88] Israel Cidon, Inder Gopal, and Shay Kutten. New models and algorithms for future networks. In *Proc. 7th ACM Symp. on Principles of Distributed Computing*, pages 74–89. ACM, August 1988.

[CGK90] I. Cidon, I.S. Gopal, and S. Kutten. Fast session setup. Unpublished Manuscript, 1990.

[CGS90] I. Cidon, I. Gopal, and A. Segall. Fast connection establishemnt in high speed networks. Technical Report RC 15515 (no. 69016), IBM T.J. Watson Research Center, Yorktown Heights, NY, February 1990.

[Eve79] Shimon Even. *Graph Algorithms.* Computer Science Press, 1979.

[GHS83] Robert G. Gallager, Pierre A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. on Programming Lang. and Syst.*, 5(1):66–77, January 1983.

[HK89] Amir Herzberg and Shay Kutten. Fast isolation of arbitrary forwarding-faults. In *Proceedings of the 8th ACM PODC*, August 1989.

[Kil87] U. Killat. Asynchrone zeitvielfachubermittlung fur breitbandnetze. *Nachrichtentech Z.*, 40 No. 8, 1987.

[KMS87] P. Kaiser, J. Midwinter, and S. Shimada. Status and future trends in terrestrial opical fiber systems in north america, europe, and japan. *IEEE Communications Magazine*, 25 No. 10, October 1987.

[MRR80] John McQuillan, Ira Richer, and Eric Rosen. The new routing algorithm for the arpanet. *IEEE Trans. on Commun.*, 28(5):711–719, May 1980.

[SLCG89] M. Sidi, W.Z. Liu, I. Cidon, and I. Gopal. Congestion avoidance through input rate regulation. In *Proc. GLOBCOM 89, Dallas Texas*, 1989.

[Tur86] J. Turner. New directions in communications, or which way to the information age? *IEEE Communications Magazine*, 24(10), October 1986.