

# The OPENET Architecture

Israel Cidon  
Tony Hsiao  
Asad Khamisy  
Abhay Parekh  
Raphael Rom  
Moshe Sidi

SMLI TR-95-37

December 1995

## Abstract:

ATM networks will soon be moving from the experimental stage of test-beds to a commercial state where production networks are deployed and operated. The progress of ATM networks appears to be at risk due to the lack of a universal, open, and efficient ATM network control platform. The emerging Private Network to Network Interface (PNNI) standard introduces a control platform that can be used as an internetwork and possibly as an intra-network solution. However, the current PNNI still falls short in providing an acceptable universal solution, due to lack of performance optimizations for intra-network operation, limited functionality, and the lack of open interfaces for future functional extensions and services.

OPENET is a common portable, open, and high-performance network control platform based on performance and functional enhancements to the PNNI standard. It is vendor-independent, scalable (in terms of network size and volume of calls), high-performance (in terms of call processing latency and throughput), and extensible (in terms of integrating customer-specific and value-added services). OPENET is designed as an extension to current PNNI so it can serve as a next generation PNNI. It is compatible with PNNI in the internetworking environment allowing large networks to be partitioned according to natural topological or organizational boundaries rather than the artificial use of internetwork interfaces at vendor boundaries.

This report describes the OPENET architecture. The major novelties of the OPENET architecture compared to the current PNNI are: the use of native ATM switching for the dissemination of utilization updates; lightweight call setup; take down and modification signaling; a new signaling paradigm that better supports fast reservation and multicast services; and a rich signaling infrastructure that enables the development of augmented services (such as mobility, directory, etc.), leveraging the existing functions of the network control platform.

 *Sun Microsystems*  
*Laboratories*

M/S 29-01  
2550 Garcia Avenue  
Mountain View, CA 94043

**email address:**  
raphael.rom@eng.sun.com

© Copyright 1995 Sun Microsystems, Inc. The SML Technical Report Series is published by Sun Microsystems Laboratories, a division of Sun Microsystems, Inc. Printed in U.S.A.

Unlimited copying without fee is permitted provided that the copies are not made nor distributed for direct commercial advantage, and credit to the source is given. Otherwise, no part of this work covered by copyright hereon may be reproduced in any form or by any means graphic, electronic, or mechanical, including photocopying, recording, taping, or storage in an information retrieval system, without the prior written permission of the copyright owner.

#### TRADEMARKS

Sun, Sun Microsystems, and the Sun logo are trademarks or registered trademarks of Sun Microsystems, Inc. UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd. All SPARC trademarks, including the SCD Compliant Logo, are trademarks or registered trademarks of SPARC International, Inc. SPARCstation, SPARCserver, SPARCengine, SPARCworks, and SPARCcompiler are licensed exclusively to Sun Microsystems, Inc. All other product names mentioned herein are the trademarks of their respective owners.

For information regarding the SML Technical Report Series, contact Jeanie Treichel, Editor-in-Chief <jeanie.treichel@eng.sun.com>.  
For distribution issues, contact Amy Tashbook Hall, Assistant Editor <amy.hall@eng.sun.com>.

# The OPENET Architecture

---

**Israel Cidon, Tony Hsiao, Asad Khamisy,  
Abhay Parekh, Raphael Rom, and Moshe Sidi**

## 1.0 Introduction

---

Network control is the set of mechanisms and processes that handles the routing, establishment, rejection, modification, maintenance and termination of connections. The actual information that is exchanged among end-users is transparent to network control. Efficient network control is critically important in the performance of high-speed real-time applications, and is a key factor in justifying the cost of Asynchronous Transfer Mode (ATM). New interactive applications (such as World Wide Web [WWW] browsers) will open many connections of different types such as text, image, video, and audio, as will future client/server exchanges. The fast handling of a large number of connections is likely to be essential to the success of ATM. Another important requirement of network control is that it enables the network to operate at high levels of efficiency, so that given a topology and link capacities, many calls can be supported with acceptable levels of service quality. A close look at network control architectures reveals that low-level functions such as the dissemination of topology and utilization updates are extremely important in ensuring efficient and reliable network control.

ATM networks will soon be moving from the experimental stage of test-beds to a commercial state where production networks are deployed and operated. The progress of ATM networks appears to be at risk due to the lack of a universal, open, and efficient ATM network control platform. The reason for this is outlined in the following subsections in which we describe the interoperability problems facing a private corporate network designer, which practically inhibit the creation of a multi-vendor ATM network and the development of necessary, customer-specific functions. The emerging Private Network to Network Interface (PNNI) standard introduces a control platform that can be used as an internetwork and possibly as an intra-network solution. However, the current PNNI still falls short in providing an acceptable universal solution, due to lack of performance optimizations for intra-network operation, limited functionality, and the lack of open interfaces for future functional extensions and services. This brings many leading ATM vendors to select their proprietary network control platform over PNNI for the task of intra-network control. The current situation poses risks for the ATM network owners, and consequently, slows the rate of acceptance of ATM technology.

OPENET is a common, open, and high-performance network control platform based on performance and functional enhancements to the PNNI standard. It is designed to address the issues of interoperability, high performance, and functionality. In addition, OPENET provides the ability to efficiently integrate customer-specific and switch-independent network control functions and value added services.

The rest of this section establishes a set of criteria by which the quality of any network control platform should be evaluated. In Section 2.0, we argue that proprietary or closed platforms are inherently inadequate in meeting this list of criteria. We also discuss the limitations of the current PNNI standard as a network control platform supporting multi-vendor or single vendor solutions. Having made the case for a universal, open platform, we then describe the OPENET network model in Section 3.0, upon which the rest of the document is based. Section 4.0 deals with novel mechanisms used in OPENET to forward control messages. Section 5.0 contains a detailed description of the signaling protocols. The topology and utilization update mechanisms are explained in Section 6.0. Route computation algorithms are also contained here.

## **1.1 Evaluation criteria for network control platforms**

The quality of the network control platform should be measured by its *performance*, *reliability*, and *functionality*.

Performance is measured by three basic quantities:

- Network control throughput, which describes the maximal rate at which the network can process new calls or change the parameters associated with existing calls.
- Call processing latency, which expresses the time elapsed between the call process request and its completion.
- Network control efficiency, which expresses how efficiently the network resources can be utilized by calls. Network efficiency is not necessarily expressed as a single numerical quantity. It is a quantitative measure of how good the routing, call admission, and call rejection mechanism operates, and how frequent calls are rejected or limited despite the fact that they could be supported.

The network control reliability is judged by the way faults (such as link and nodal outage, control message errors, and soft errors) are handled. Since the ATM network control actually reserves resources such as bandwidth and VC numbers, unused resources must always be returned after call termination (including call termination that is forced by an outage). Also, while the ATM infrastructure is perceived as high-speed and reliable, the large number of connections and their rapid changes raise the risk of gradual degradation due to resource deadlocks and livelocks, similar to operating systems which allocate memory and other resources within a computer.

The network control functionality is judged by the level of service that the network control provides to users. Functions such as mobile naming support, QoS negotiation, supporting multi-

level of call priority, third party setup, policy routing and many more, may be needed by particular network owners. For example, a company that provides a personal (mobile) communication service via its private ATM terrestrial infrastructure must include on-line efficient tracking of mobile users and hand-off mechanisms as part of its network control. Other companies may not need such a function. This calls for the ability to add functions to the network control according to the network use and needs, and highlights the advantage of having an open interface through which such functions can be added to the network control.

## **2.0 The Case for an Open Network Control Platform**

---

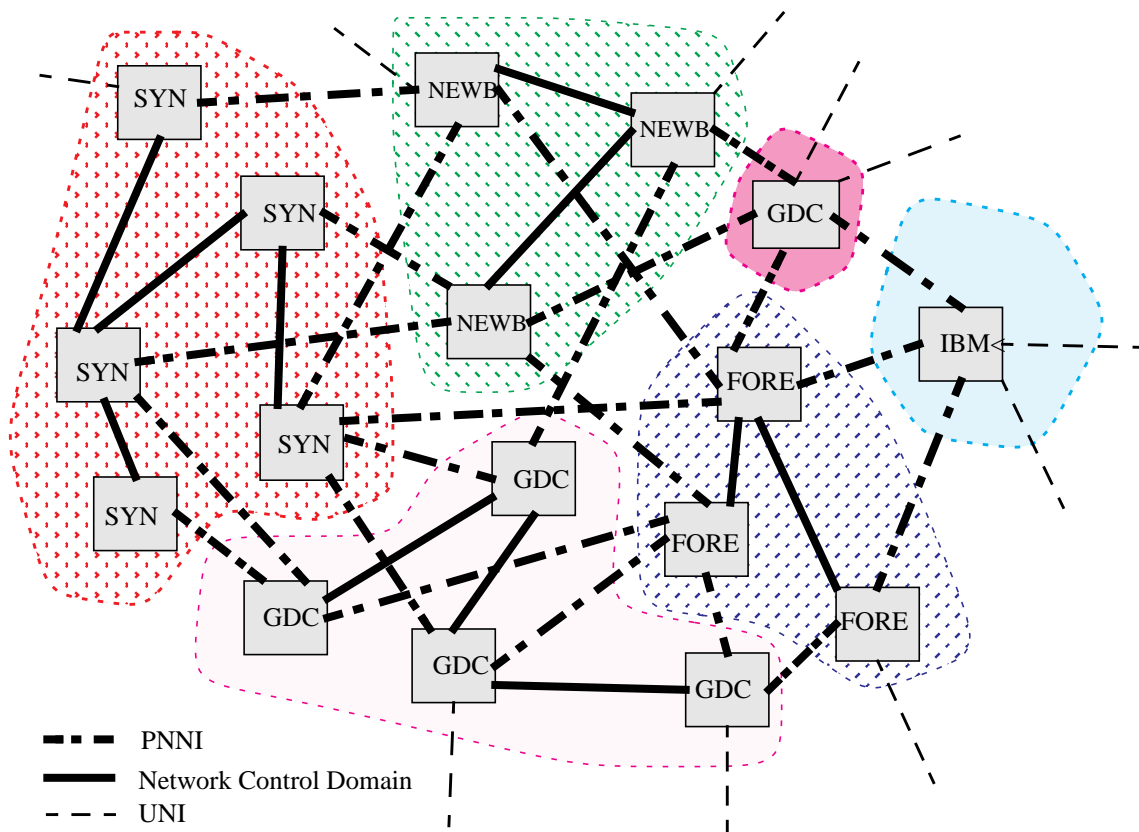
A major inhibitor to the evolution of ATM networks from experimental test-beds to the commercial world with demanding performance and reliability requirements is the lack of a universal, open, and efficient network control platform. The interoperability problems facing a private corporate network designer practically preclude the creation of multi-vendor ATM networks, leaving open the risky option of building single vendor networks. In addition, it may be extremely difficult, if not impossible, to support the development of necessary, customer-specific functions such as custom billing, directory services, and mobility management. In this section, we will demonstrate that an open network control platform, such as OPENET, does not suffer from these drawbacks, and in fact, provides significant added value over proprietary solutions.

### **2.1 Multi-vendor campus area networks**

Figure 1 depicts a single campus area network composed of a large selection of ATM switches from multiple vendors. We will explain later why it is necessary to build such a multi-vendor network (or at least have the ability to build such a network). Different vendor switches are clustered together to form different network control domains or islands. These domains were not created to solve network scale problems, but are more or less an arbitrary partition of the network due to artificial vendor boundaries. It is easy to realize that unless a special effort is made at the time of the design, most links of the network are inter-domain links, while only few are intra-domain. This by itself implies that the importance of the routing efficiency of intra-domain network control is less significant than the efficiency of the inter-domain mechanism in this scenario.

The current situation is that each domain is controlled by proprietary software that varies tremendously among vendors. The different domains are characterized by incompatible signaling, algorithms, data structures, and functions. These will have to be translated each time a domain is crossed, possibly at almost all links.

The multi-vendor control platforms lack common design objectives, and therefore might possess conflicting properties. For example, each specific design must trade-off among various design objectives that compete in terms of network control functions, computing power, and the design effort. Such objectives could include: speed of call setup and modification, quality



**Figure 1.** Multi-Vendor ATM Network

and optimality of computed routes, software scalability (in terms of maximal number of nodes in the network, and the maximal number of calls supported per node or link), fast recovery after topological changes, optimization of bandwidth usage, optimization of QoS conformance, cost (reduced memory size and processing power), support of advanced functions such as transparent route switching, naming directories or call priority handling, etc. Each vendor selects a particular compromise among the different objectives, stressing the ones they believe are the key to their customers. However, when a multi-vendor network is constructed, each objective will be met at the least common denominator. Calls will have to cross multiple domains, and most supported functions will cease to exist because of a lack of universal support.

Quality of service may also have different interpretations at each domain. With the lack of a common “glue” among the different vendor islands, the customer should assume the worst case in terms of quality of service as well as the worst case in terms of bandwidth usage efficiency.

It is assumed that the inter-domain protocol will be the set of PNNI signals which are currently under development at the ATM Forum [3]. PNNI appears to be designed and optimized for an *internetworking* protocol—a set of conventions that is employed when two different (by choice) networks (we will term them subnetworks) are to be connected. It is likely that many vendors will use it exclusively as an inter-domain “glue” between their proprietary “islands.” (We will deal later with the scenario in which PNNI is used as an intra-networking protocol.) The fact that PNNI is designed as an internetworking solution has major consequences in terms of overall network control performance in such scenarios: there must be multiple and expensive translations of the proprietary data-structures and signaling into the external standard signals each time a vendor domain is crossed. This causes a major overhead penalty, as well as information loss because of inability of the PNNI to support all the possible information types and precisions used by all proprietary platforms. While many aspects of the PNNI standard are well suited to tackling this problem among natural and logical network partitions, we believe it falls short in solving among the arbitrary vendor-based partitions.

Since the PNNI design is optimized to scale well with the size of the network, its main objective is to restrict the amount of information flow between the sub-networks. It therefore defines a hierarchical description of topologies in which the topology description of individual subnetworks are collapsed to a single node or some other highly compressed structures (such as trees). This implies that feasibility and optimality of routes are sometimes lost. In addition, the call setup procedure must be conducted in two phases (rather than one); first the routing at the domain level will be conducted by the source over the compressed (higher hierarchy picture) view of the internetwork, and then each domain (in sequence) will need to compute an internal path. This is likely to cause an excessive delay compared to the same computation in a single-vendor undivided network.

Finally, the maintenance and problem-tracking in such a network will be extremely difficult to handle. The set of proprietary network control platforms and the PNNI gluing them together will be complex to understand or track. Since many of the functions are automated (the routing and setup of calls), adaptive, and extremely frequent, it will be hard to isolate the network control or network control boundary that is responsible for a failure such as a high rate of call blocking or for improper overall routing. The maintenance of the software and its upgrades at the various domains will also become a very hard task for the customer who struggles with a completely new, and to a large extent exploratory, technology.

Another possibility is for certain vendors to adopt PNNI as both an intra-domain and inter-domain solution. However, a quick look at current PNNI reveals major performance and functional limitations which are caused by the fact that PNNI is optimized for inter-domain environments, and because of the wish to develop it very quickly, thereby relying on existing solutions (such as OSPF and UNI signaling) that were not designed or optimized for the switch-to-switch ATM environment. Current PNNI is based on extensive flooding (similarly to OSPF [9]) of topology and utilization information using relatively large packets (250-300 bytes for every link state update). Such a flood will result either in excessive burden on the network control processors or in a severe limitation of the speed and accuracy of information update, and hence a degradation of the routing efficiency. Similarly, PNNI uses extended UNI

signaling between switches. UNI in turn is based on the ISDN signaling protocol suite built over a rather complex reliable data-link protocol (SSCOP - Service Specific Connection Oriented Protocol). The low throughput and high latency of such a call signaling mechanism will severely limit the network control performance. Since certain ATM vendors have already implemented better performance network control platforms, it is unlikely that they will adopt the current PNNI as their preferred intra-domain network control. Currently, most performance-sensitive functions of the current PNNI do not have any clear hooks for future hardware assist, in contrast to other network control designs such as PARIS [8]. Therefore, the current level of performance is not expected to be dramatically improved in the future. PNNI lacks a publicized open interface, which makes it difficult for switch vendors to incorporate extensions and advanced features that they believe should be provided.

## **2.2 Single-vendor campus networks**

Single-vendor solutions are quite typical in the router market (in particular the pre-OSPF router market). However, we believe that the situation is quite different when it comes to ATM networks.

The router market developed gradually out of growth demands of existing networks, and thus was a necessity for a large base of users who were already highly invested in LAN technology. ATM, on the other hand, offers completely new technology from LAN to WAN. The high expectations that surrounded the BISDN market led to the existence of dozens of ATM providers, all competing for a market that does not currently exist. This is clearly a transition period with everybody expecting that only a handful of market leaders will eventually be established. However, no such market leaders are established today. This in turn, slows the establishment of production networks which provide the experience necessary for the ATM technology evolution and selection process.

In addition to the “chicken and egg” problem that stalls the market and prevents customers from choosing their single vendor (locking them in for a fairly long time), there are many open technical issues. The ATM switch technology is largely immature. The customer is constantly confused by conflicting statements about the value of non-blocking switches, expandable switch designs (in terms of number of ports as well as port speeds), value of QoS support within the switch fabric, amount of buffers within and outside the fabric, input and output queueing, buffering policies, the required amount of multicast support, etc. It seems that eventually only a few of these architectures and options will prove to be the best choice. The customer is still confused regarding traffic management schemes and whether there will be differences among the different vendors. Network control is also a major issue. Most vendors cannot offer full network control or evaluate its network control performance, reliability and functionality. Customers cannot understand, at this point, whether their selected vendor will be able to lead (or survive) in this open issue as well.

The last problem is that of specialization. Unlike the router market, the ATM market will probably have a large spectrum of products, sometimes very different in internal technology and price/performance/reliability objectives. For example, there are pure ATM basic access

switches (hubs), high speed campus interconnect switches (with a possibly different range of link speeds), ATM protocol converters (in particular legacy LANs to ATM), public network access equipment, and backbone WAN switches. There are very few vendors that provide the entire equipment spectrum and it is unclear whether or not they will become market leaders at all market segments.

Since most corporate customers already own existing private networks they are not forced to move to ATM, and will probably avoid the risk currently involved in the transition. Meanwhile, they continue to invest in their current networks, making the future transition even more difficult. New breeds of switching hubs and routers, as well as new data networking architectures and standards might evolve to make ATM less and less of a necessity for high-speed applications. In such a scenario, ATM acceptance will be limited to the low volume, high-cost WAN and public networking where no viable alternative for ATM seems to exist.

We conclude, therefore, that the current trend in ATM network control is quite risky and might become a market inhibitor for all ATM providers and switch vendors. While both flavors of PNNI (as a glue between vendor islands or as a uniform intra-networking solution) are considered by many to be an adequate and reasonable solution, it is our belief that the reason for this misconception is the fact that the complexity and challenges of providing a high-performance, reliable, and high-functionality multi-vendor ATM network control are still largely underestimated. This is mainly because very few vendors have already fully developed, implemented, and tested their network control, and many ATM vendors have relatively little experience in developing connection-oriented network control platforms.

### **2.3 Open network control and its intrinsic customer value**

The problems posed by a complex mixture of heterogeneous, proprietary network control platforms can be fully solved by using a homogeneous and open platform. However, this mandates that the open network control (ONC) be relatively easy to port to a diverse number of platforms and be competitive with proprietary platforms that are currently being developed. Therefore, the ONC should be designed to be:

- Portable to both stand-alone or under-the-cover processors. This implies a lightweight and simple implementation, as well as being easily partitioned among multi-processors (such as the combination of link processors, a switch processor, and an external general purpose processor).
- High performance in terms of low call accept/reject latency and high rate of calls processing (establishment, change, and termination).
- Ready for future enhancements through hardware assistance to further enhance call processing throughput and latency for future requirements. This implies a low level implementation of key performance processes.
- Scalable in terms of network size (large number of switches and links) as well as a high volume of call establishment and call property changes per link or switch.

- Provide several open application programming interfaces (APIs) at various levels to enable an efficient split of functions, as well as a highly modular structure to enable easy module replacement. This will also enable many functional extensions performed by various interested parties that will run over all vendors' platforms.
- Be compatible with PNNI in the internetworking environment where large networks need to be partitioned according to natural boundaries (geographical locations, sites, buildings, organizations) or different organizations need to provide connectivity.
- Be a natural extension to current PNNI so it can be accepted as a next generation internetworking PNNI.

In addition to solving the problem of ATM market inhibitors, a uniform network control solution has many new advantages and intrinsic values to the ATM network customer. Some of these value-added advantages are:

- Freedom of choice. An open network control platform allows customers to buy the best hardware solution for their networking needs without compromising network control, efficiency and complexity.
- Lower risk. An open network control enables customers to decide on an ATM vendor based on their present judgement of the current product line without being "locked" to this vendor in the future.
- Performance. An open network control eliminates the artificial use of internetwork interfaces at vendor boundaries, allowing a single domain operation at logical boundaries. Therefore, it will result in less overhead and less hierarchy in the routing and setup process. The particular (mostly single cell) architecture proposed here results in small setup overhead, high call throughput and scalability. It is also amenable to future hardware implementation, given its low level design and its simple structure.
- Management, problem tracking, and learning. The reduced complexity and number of conversion instances between different network control platforms, as well as a small number of domain and boundary crossings enables a much more uniform management and problem tracking process. The customer needs to understand and track only a single control system.
- Leveraging existing network experience and research. Based on OSPF/PNNI concepts for topology control and open extensions for resource management, the proposed open network control platform provides a proven routing protocol and is also optimized for the rigors of resource management on a high speed network. The openness will provide an avenue for future extensions/enhancements by the research community.
- Software and applications development. Third parties will easily be able to extend OPENET with open or commercial network support applications such as billing, security, etc. Similarly, customers, in particular, large ones, can write their own specific network applications (such as directory services, security, policy-based routing and many more) above a single and open API.

### **3.0 OPENET Network Model**

---

At a high level, ATM networks can be viewed as a collection of *ATM-nodes* interconnected by high-speed *links* in a networked fashion to provide ATM service to a number of external *users*. A typical node is comprised of two main functional parts: the switch and the *Control Point* (CP). The switch contains a switching fabric (which includes buffers) and the switch control module. The switch control module controls and coordinates the operation of the buffering and switching unit. For performance reasons, it is typically implemented in hardware such as lookup tables, finite state machines, etc. The CP is responsible for all other control functions such as initializing and maintaining the switching tables, coordinating activities with other switches and with network management facilities, and serving the individual user requests. The CP is essentially a logical entity (or a process) which is typically implemented in software, although firmware and other software/hardware configurations are also possible. The OPENET architecture focuses on CP functions.

The physical device on which the CP is implemented is termed the *Controlling Device* (CD). The CD may not be a single entity, that is, parts of the CP can be implemented in different hardware pieces. For example, each link adaptor of the switch could contain a separate micro-controller and possibly an additional central micro-processor which is responsible for the common switch functions. Another implementation option is a single CD that hosts several CPs, each controlling a different switch and possibly communicating with each other within the same CD.

Two CPs are called *neighbors* if they control two switches that are directly connected by a link.

To perform their functions, CPs need to exchange data with one another. The mechanism to achieve this is termed *control VC* which is typically implemented as a regular (permanent) VC designated for this purpose. (Such a construct is also included in the PNNI and UNI 4.0 specification.)

We assume that every switch in the network possesses a 6byte ID, which is guaranteed to be unique within the network. This conforms to the PNNI assumptions as well. While PNNI refers to a switch using a 22 byte address that includes the same 6 byte ID plus a network prefix, OPENET messages will generally use the 6 bytes field in as much as possible order to reduce overhead.

Links have IDs that are local to the switch from which they emanate. Links are locally identified by the local port number kept at the switch. This allows for short local link IDs. Therefore, to globally identify a link, a combination of a node ID and port ID is used. Sequences of Port IDs appear in the messages that are exchanged between CPs, and hence, it behooves us to choose as short a descriptor as possible. On the other hand, choosing too short a descriptor may render the architecture too limiting in terms of scale. We believe that 2 bytes suffice to describe port IDs, since we envision 64K as a practical upper bound on the number of links per switch.

Users are identified by a user-ID that is taken from the ATM name space. Within the network, users are identified by the port that connects them to the network, similar to links between switches. There is, thus, from the CPs' point of view, a 1-1 correspondence between the user and its attachment point.

The current OPENET architecture does not require a CP to be a uniquely identifiable entity within the network. That is, CPs include in their messages the relevant switch ID in their cluster, and switches should identify cells destined to their CP (say these messages carried over particular control VCs) and forward them appropriately. This is the mechanism by which control messages are exchanged between CPs. It might be necessary in the future to distinguish CPs from switches in order to allow a dynamic association of CPs and switches for fault-tolerance or computation load-balancing. However, such functions are not yet part of the OPENET architecture.

### **3.1 CP functions**

In general, CPs perform those functions that are necessary to make a group of switches operate as an integrated network. The process of integration entails understanding of and compensation for the differences between switches of different makes and models. In this section we briefly delineate several of these functions; a more detailed description appears later in this document.

**User Support:** The CP is responsible for interpreting user requests, received via the UNI interface, and translating them into individual actions that together provide the user the service it requested. This function requires that all control information issued by the user (typically carried over VC #5) is diverted by the entry switch to its CP for interpretation and action. At this time, user support is dictated (and limited) by the current ATM standard (see the ATM Forum's UNI specifications [1]). As it stands today, users can request a quality of service while providing a description of traffic intensity for that class. This request is presented to the "network," namely the source CP, which must map the request to the internal representation of the network control. We have adopted the PNNI view of QoS in terms of how a source and a resource can specify their QoS request or guarantee.

Users are also limited by the type of service they request, due to the limited functions supported by the ATM Forum's UNI. However, we expect that this will change and a new UNI version will specify new services. We have decided to extend the set of functions supported by OPENET beyond UNI 4.0; in particular where the PNNI or switch-to-switch signaling must be extended to support new and attractive services. Detailed descriptions of the following functions are given in Section 5.0.

**VC set-up:** The most basic user request is establishing a connection between the user and a specified destination according to a certain performance and quality of service specification. Once such a request is received at a CP from the user, it is mapped into internal network parameters, and an attempt to establish the connection is done. To service such a request, every CP must accurately know the current local allocation of resources (e.g., allocated bandwidth and

buffers). Additional information about the network topology and current link utilizations can tremendously facilitate the setup process. To that end, every CP maintains a view of the network (both topology and utilization) which enables it to calculate a complete provisional route that then must be verified.

A special type of VC is the multicast VC. This is a virtual circuit with a single source and multiple destinations. Such VCs are typically organized in a tree structure (as opposed to a “shoe string” structure of a regular VC) and hence, presents further difficulties in its set up.

OPENET provides several new useful setup functions that are not yet supported by UNI 4.0. It allows the establishment of both unicast and multicast VCs from a point which is away from the source (third party setup). It allows the computation and establishment of a complete multicast tree as a single operation, whereas ATM today only supports adding connections one at a time. OPENET also provides new multicast connections (termed distribution trees) which allow multiple sources and multiple destinations. At this point, these connections are used for the facilitation of the network control itself.

**VC take-down:** Once the VC is no longer needed (e.g., as a result of user indication or some failure), an efficient and orderly procedure for taking it down is necessary. The procedure must be efficient so that all the resources that were allocated to the VC can be reclaimed by the nodes. This is required so that these reclaimed resources are available for reallocation as soon as possible.

**VC maintenance:** Once a VC has been established and data has started flowing through it, the operation of the VC must be maintained. These functions typically deal with managing the allocated resources in the face of changing circumstances during the VCs lifetime. For example, a user having made a reservation may find it necessary to change it (increase, decrease, or otherwise alter the VCs parameters). In addition, changes in network topology or link utilization necessitate reconfiguring VCs while in operation. It is also useful to provide refresh mechanisms to check for the liveness of connections.

**Topology update and leader election:** Our architecture (following the ATM/PNNI specifications and other connection-oriented legacy networks), calls for a computation of the complete VC path at the source. To be able to construct feasible routes, a node must maintain information regarding network topology. Therefore, every node maintains an image of the complete network; that is, every node recognizes the nodes and links that are operational at any given time. Because a network is a distributed system, this information cannot be accurate at all times, but is accurate up to possibly the most recent update. As the private ATM system is typically built around a high-speed, optical backbone infrastructure, topological changes are expected to be infrequent, and therefore at most times the topological information in the node will be accurate. The OPENET approach is to leverage the PNNI topology update mechanism in order to simplify the implementation, maintaining compatibility and providing smooth network-to-network integration. (See Section 6.0 for details.) Similarly to PNNI, OPENET elects one of the CPs as a network leader (or peer group leader). OPENET provides a simple mes-

sage- efficient election which improves the current PNNI election by a linear (in the number of nodes) factor.

**Utilization update:** To be able to calculate efficient routes (from among the feasible ones) every node maintains a utilization image of the network, i.e., the level of bandwidth reservation of every link in the network with respect to every class of service. Unlike topological updates, utilization data change very frequently; in fact they change with the establishment, reservation change, and termination of VCs. Therefore, our mechanism for distributing the utilization information substantially differs from that of the topology update. As explained in Section 6.0, utilization updates are disseminated along a specially constructed spanning tree with unicell updates so as to operate efficiently while consuming as little resources as possible.

**Network management support:** No modern network can operate without a network management facility. Since most of the data of interest to the network manager are maintained by the CP, it is natural for the CP to support network management functions.

**Higher level application support:** OPENET provides extensive support for new services and network control extensions through open APIs. The support includes various ways for distributed applications to communicate between CPs via datagram and connection-oriented unicast, one to many multicast, and many-to-many multicast. The support also includes an open access to the OPENET data structures such as topology and utilization databases, ATM address lists, connection data, etc.

**Additional functions:** Several additional functions may be incorporated into the CP in the future. Some of these functions are necessary for network operation. Their implementation within the CP are both a matter of efficiency and functionality. Address directory service is a good example. Directory services must be provided somewhere in the network so that translation between different name spaces (in particular the ones used by ATM) takes place. Because the CP maintains the network topology it would be natural for such a function to be implemented within the CP ([10][11][12]).

### 3.2 CP data structures

Every CP maintains a version of the hardware routing table of the switch that it controls. In general, the CP tables need not be an exact replica of the hardware table, but they should contain all the information derivable from the hardware table, plus some extra information used for control only. In this document we define only an abstraction of the data structure represented in the CP's tables, since an efficient implementation of this abstract representation will depend on the actual architecture on which the CP resides. Two types of tables are defined here: the swap table and the resource table.

An *entry* in the CP's *swap table* consists of  $n$  pairs of input label/input port and  $m$  pairs of output label/output port:  $(VC_{i1}, P_{i1}), \dots, (VC_{in}, P_{in}) \rightarrow (VC_{o1}, P_{o1}), \dots, (VC_{om}, P_{om})$ . A CP with this entry will "program" the switch hardware to route any cell with input label/input port matching *one* of the  $n$  input pairs to *all* the  $m$  output ports with the corresponding label swapping according

to the given entry. Thus, a cell entering port  $P_{ij}$ , ( $1 \leq j \leq n$ ) with label  $VC_{ij}$  will be multicast to ports  $P_{o1}, \dots, P_{om}$  with corresponding swap labels  $VC_{o1}, \dots, VC_{om}$ , respectively. Note that this representation handles the general case of an  $n$  to  $m$  multicast connection. Also note that a CP's labels might correspond to actual (or physical) labels at the switch hardware, or they might be used for control only without such a correspondence (these are referred to as *logical* labels). In other words, CPs can maintain structures with no actual equivalent VC established at the switch hardware using logical labels. In this case, it is recommended that label values outside the range of the hardware switch VC labels be used in order to avoid resource waste.

The assignment of port numbers is done by the CPs, either independently or in coordination with the next node along the connection (as will be explained later in this document). By convention, the local CP is always designated as port 0 (and is not shown in the examples and figures of this document).

A given entry in the CP's table represents a *direction* in which cells or messages will travel. Hence, a single entry will be sufficient to represent a unidirectional connection. For bidirectional connections, two separate entries in the table are needed. We say that an entry  $ENT_i$  in the table is *associated* with entry  $ENT_j$  if they represent opposite directions of the same connection in a bidirectional connection. In general, an *association* between entries  $ENT_1, \dots, ENT_k$  specifies a relation between the entries, such as being the opposite directions of the same connection, or being part of a distribution tree, etc. We denote this relationship by  $ENT_1 \leftrightarrow \dots \leftrightarrow ENT_k$ . In other words, associated entries in a VC table are closely related entities which should be stored in a way that given an entry, one can easily locate the other entry(ies) within the table belonging to the same "association."

In addition to the VC label information, a CP must maintain the resource utilization record of each connection that traverses its switch. The CP maintains for each output port and class of service a *resource usage table* with entries consisting of the label associated with a connection and the resources allocated for that connection. Also, to support the ABR flow control mechanism the switch and CP must maintain per-VC information which is also stored in this VC table. The set of resources are described in the PNNI specification.

### **3.3 Connection types**

User and control information is carried over connections. While ATM distinguishes between VPs and VCs, we will limit our discussion to VC-based switches (while similar discussion can be conducted for VPs as well).

The current ATM standard recognizes two types of connections. The first is point-to-point connection (also termed 1-to-1 or unicast) where a single source (or calling party in the ATM terminology) exchanges information with a single destination (the called party). All point-to-point connections in ATM are considered bidirectional (information may flow at both directions). The ATM switch performs switching and label swapping functions which follow the internal structure of the switch. We will describe these functions with the assistance of the generic VC table as described in Section 3.2.

The second type of connection currently supported by the ATM standards is a connection over which a single source may send the same stream of cells to multiple destinations. This connection is termed unidirectional multicast (or 1-to-n multicast, point to multi-point). Such a connection is established by maintaining a directed tree structure (termed a *destination tree*) over which cells are being sent from the root to the leaves. At every junction in the tree the switch must copy the same cells to multiple output ports and perform a VC label swap operation specific to each output. This information will be maintained in the switch as described in Section 3.2. Both of the above connection types permit only a single cell source to be multiplexed at a receiver. Therefore, there is no problem with fragmenting and reassembling larger source data units using one of the available adaptation layers.

In [4], another possible structure we term a *source tree* is described. This connection carries multiple streams of cells from multiple sources to a common destination. This tree is directed toward a single destination, which means that at every junction node, a group of one or more port IDs ( $P_{in}$ ) and input VC labels ( $VC_{in}$ ) should be mapped into a single output port ID ( $P_{out}$ ) and output VC label ( $VC_{out}$ ) (see Section 3.2). A source tree multiplexes cells of different sources at a single destination which receives all of them with the same VC label. This implies that an adaptation layer that relies on the VC value only cannot be applied (such as AAL5) to a source tree. Therefore, the tree can be used to deliver unicell messages, or supports an adaptation layer which makes use of several of the 48 data bytes portion of the cell (such as AAL3/4).

A *distribution tree* is a multicast construct with several sources and several destinations. The number and identity of the sources and destinations does not have to be identical. A cell sent by any of the sources on the distribution tree will arrive at all the designated destinations. Similar to the destination tree case, a distribution tree may be used for unicell messages or under adaptation layers that allow the multiplexing of several sources over the same VC label. In the following, we describe two approaches to establishing a distribution tree, namely, the load-balanced tree and the pivot-based approaches.

### 3.3.1 The load-balanced tree approach

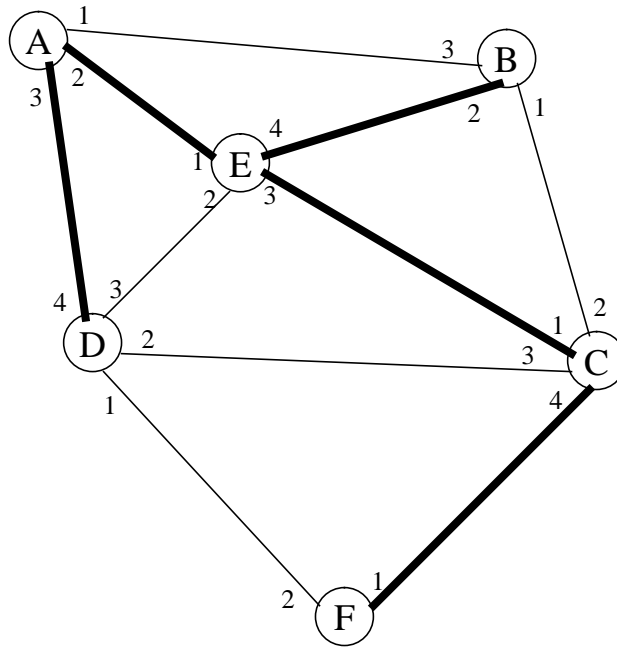
A tree spanning all of the sources and destinations is constructed so that each link of the tree is bidirectional. Thus, two VCIs are clearly defined between any two neighboring nodes of the distribution tree—one in each direction. Cells are then broadcast on this tree, i.e., each node associates every incoming VCI of the destination tree with all the outgoing VCIs on the links of the distribution tree *except* in the return direction (i.e., excluding the link of the incoming VCI).<sup>1</sup>

---

1. In the case in which some of the nodes are exclusively sources or destinations, communication can be reduced further by refining the tree as follows: delete all paths originating at a node that is a destination but not a source, up to the first node that is also a destination; and similarly, delete all paths that consist of links that are oriented into nodes that are source nodes but not destination nodes.

### 3.3.1.1 A Detailed example of the load-balanced tree approach

As an example, consider the network of Figure 2. In the example, the circles are nodes whose names are marked with upper-case letters, the bold lines are the links of the designated tree, and the port numbers are marked with numerals. (as mentioned in Section 3.2, the assignment of port numbers is done locally and arbitrarily, and by convention, the local CP is always designated as port 0 and not shown in the figure). The VC tables of all the nodes appear in Table 1. For example, a cell injected to the network at node C (port 0) is switched to both port 4 (towards node F) with a VCI of 18, and port 1 (towards node E) with a VCI of 30. This cell, upon arrival at node E (on port 3) with that VCI will be forwarded appropriately towards nodes A and B and towards node E's own CP. As is evident from the example, every cell will traverse every link exactly once, thereby balancing the load on the links of the tree.



**Figure 2.** An example of a utilization tree setup

### 3.3.2 A pivot-based approach

In some cases, it may be beneficial to set up the distribution tree about a specially designated node called the pivot. Each source forwards all of its cells to the pivot, which then forwards these cells to each destination. Thus, the pivot-based tree is composed of a combination of a source tree with a destination tree that share the same node as the root of both trees. A special mapping of the VC labels directs the traffic multiplexed from all sources (over the source tree) to be multicast over the destination tree to all its destinations.

Consider a set of sources  $s_1, s_2, \dots, s_n$  and a set of destinations  $d_1, d_2, \dots, d_m$ . Choose a *pivot* switch  $p$  (which can be any switch) and construct two directed trees, one consisting of the switch  $p$  along with all the  $s_i$  switches (the “source” tree), and the other consisting of the

Input Designation		Output Designation			
<i>P</i>	<i>VCI</i>	<i>P</i>	<i>VCI</i>	<i>P</i>	<i>VCI</i>
0	25	2	23	3	24
3	18	2	23	0	24
2	21	3	24	0	24

**VC Table of Node A**

Input Designation		Output Designation	
<i>P</i>	<i>VCI</i>	<i>P</i>	<i>VCI</i>
0	20	2	27
2	23	0	20

**VC Table of Node B**

Input Designation		Output Designation			
<i>P</i>	<i>VCI</i>	<i>P</i>	<i>VCI</i>	<i>P</i>	<i>VCI</i>
0	13	1	30	4	18
4	31	1	30	0	18
1	22	0	18	4	18

**VC Table of Node C**

Input Designation		Output Designation	
<i>P</i>	<i>VCI</i>	<i>P</i>	<i>VCI</i>
0	20	4	18
4	24	0	11

**VC Table of Node D**

Input Designation		Output Designation					
<i>P</i>	<i>VCI</i>	<i>P</i>	<i>VCI</i>	<i>P</i>	<i>VCI</i>	<i>P</i>	<i>VCI</i>
0	19	1	21	3	22	4	23
3	30	1	21	0	12	4	23
4	27	1	21	3	22	0	12
1	23	0	12	3	22	4	23

**VC Table of Node E**

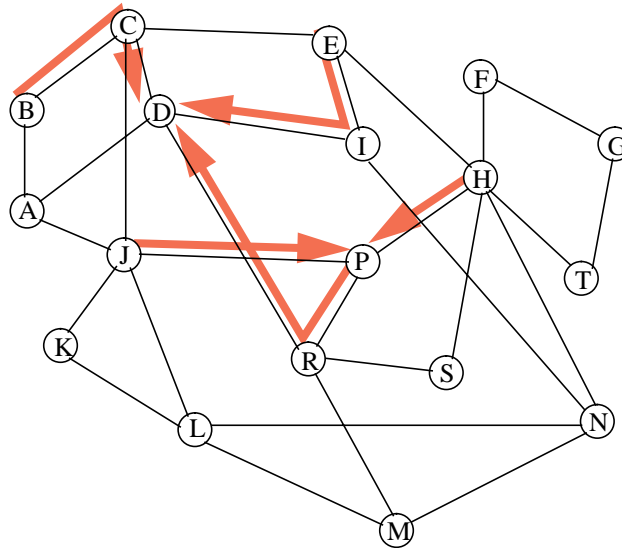
Input Designation		Output Designation	
<i>P</i>	<i>VCI</i>	<i>P</i>	<i>VCI</i>
0	31	1	31
1	18	0	12

**VC Table of Node F**

**Table 1.** VC Tables for the load-balanced tree example

switch  $p$  with all the  $d_i$  switches (the “destination tree”). The source tree is directed towards  $p$ , and the destination tree is directed away from  $p$ . The construction of the trees is done according to the normal way of VC construction. Thus, a cell entering at any of the switches  $s_i$  carrying the VCI of the source tree will end up at the pivot switch. Using the multicast capability, every cell entering the pivot with the VC identifier of the destination tree will be switched and distributed to all destinations  $d_i$ . By appropriately setting the VC-table at the pivot so that the source tree VCI and the destination tree VCI are linked together, the distribution tree is constructed.

To be more concrete, Figures 3, 4, and 5 show the construction of such a tree. Suppose it is required to build a distribution tree where the sources are nodes  $\{B, E, H, J, R\}$  and the set of destinations is  $\{C, F, J, L, M, T\}$ . We choose node  $D$  as the pivot and construct a source tree as shown in Figure 3. The source tree requires a single VCI per link in each of the switches that constitute the tree. Any cell inserted at one of the source nodes ends up at the pivot,  $D$ . Similarly, a destination (multicast) tree is constructed as shown in Figure 4. Again, a single VCI per node is required to maintain that tree, and any cell inserted at the pivot  $D$  will arrive at any of the prescribed destinations. The combined source-destination tree is depicted in Figure 5, completing the construction of the distribution tree.



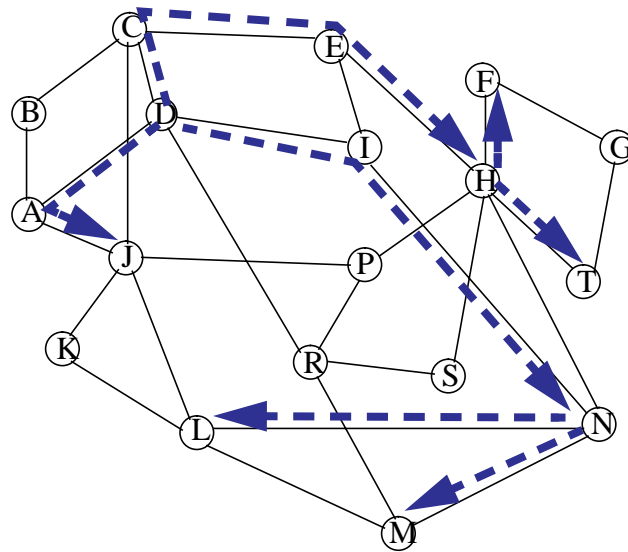
**Figure 3.** The source tree

### 3.3.2.1 A Detailed Example Of A Pivot Based Approach

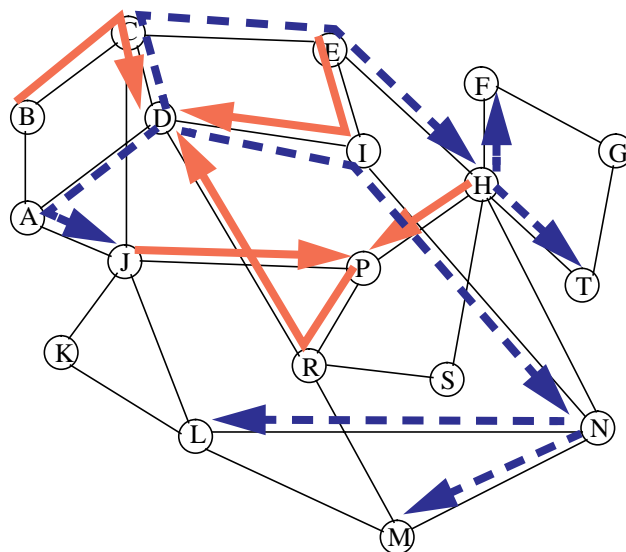
In some examples such as the utilization update where all nodes of the networks are involved, the entire set of nodes are both sources and destinations. In such a case it may be useful (though not necessary) to construct a single tree that serves both as the source and destination trees. Figure 2 shows an example of such a network. Node  $E$  is the pivot in this example.

A cell arriving at an input port with a certain VCI is switched to the associated output port with the associated VCI. If multiple output designations appear the cell is switched on all the designated output ports each with its own (possibly different) output VCI. Note that this is only an abstract notation for explanation purposes; the specific implementation is switch dependent. Tables 2 and 3 depict these tables for the pivot and regular nodes.

Thus, if a cell injected by the CP of node  $D$  (on port 0) with a VCI of 20 will be switched out on port 4 with a VCI 18. This cell will arrive at node  $A$  on port 3 and hence, will be switched out on port 2 with VCI 23. This cell will then arrive at node  $E$  (the pivot) on port 1 and will be switched out on port 1 (towards node  $A$ ) with a VCI 21, on port 4 (towards node  $B$ ) with a VCI 23; on port 3 (towards node  $C$ ) with a VCI 22; and of course also to port 0 (node  $E$ 's own CP).



**Figure 4.** The destination tree



**Figure 5.** Combined source and destination tree

Traversing these cells according to the table shows that this cell will arrive at all CPs in the network.

A drawback of the pivot-based approach over the load-balanced tree approach is that the traffic load on the links is not completely balanced. For instance, in the example above, if node C

Input Designation		Output Designation							
<i>P</i>	<i>VCI</i>	<i>P</i>	<i>VCI</i>	<i>P</i>	<i>VCI</i>	<i>P</i>	<i>VCI</i>	<i>P</i>	<i>VCI</i>
0	19	1	21	3	22	4	23	--	--
3	30	1	21	3	22	4	23	0	12
4	27	1	21	3	22	4	23	0	12
1	23	1	21	3	22	4	23	0	12

**VC Table of Node E**

**Table 2.** VC table for the pivot node

Input Designation		Output Designation			
<i>P</i>	<i>VCI</i>	<i>P</i>	<i>VCI</i>	<i>P</i>	<i>VCI</i>
0	25	2	23	--	--
3	18	2	23	--	--
2	21	3	24	0	24

**VC Table of Node A**

Input Designation		Output Designation	
<i>P</i>	<i>VCI</i>	<i>P</i>	<i>VCI</i>
0	20	2	27
2	23	0	20

**VC Table of Node B**

Input Designation		Output Designation			
<i>P</i>	<i>VCI</i>	<i>P</i>	<i>VCI</i>	<i>P</i>	<i>VCI</i>
0	13	1	30	--	--
4	31	1	30	--	--
1	22	0	18	4	18

**VC Table of Node C**

Input Designation		Output Designation	
<i>P</i>	<i>VCI</i>	<i>P</i>	<i>VCI</i>
0	20	4	18
4	24	0	11

**VC Table of Node D**

Input Designation		Output Designation	
<i>P</i>	<i>VCI</i>	<i>P</i>	<i>VCI</i>
0	31	1	31
1	18	0	12

**VC Table of Node F**

**Table 3.** VC tables of regular nodes

transmits a cell it will wind its way to node E (the pivot) and back towards node F, thus traversing the link CE twice, once in each direction.

## 4.0 Forwarding Modes

---

OPENET defines two mechanisms, or forwarding modes, in which CPs forward control and other messages to other CPs. The *source route* and *VC traversal* forwarding modes described in this section allow a CP of, say, switch  $s_1$  to forward messages to the CPs of switches  $s_2, \dots, s_n$  in a given order. In the source route forwarding mode, a complete path is given in the message which allows the CP to forward the message, whereas in the VC traversal forwarding mode, the message travels along the same path as an established VC. The messages are forwarded between neighboring CPs over the pre-established control VCs (see Section 3.0). OPENET supports a mechanism for exchange of VC labels which allows different labels to be used for opposite directions of a connection. This mechanism is more general than the mechanism supported by UNI 4.0 and PNNI 1.0, and hence, it can be readily specialized to conform to UNI and PNNI if necessary.

### 4.1 Source route forwarding mode

In the source route forwarding mode, a message traverses a path from CP to CP according to a route that is explicitly contained within the message. Each CP along the path not only forwards the message to the next CP along the path, but also performs a certain function that is also indicated in the message. Such a message is used, for example, by the VC set-up mechanism (see Section 5.1) in which the typical function would be bandwidth reservation. It is worthy to note that given a source node, a path can be uniquely described by a sequence of local link/port IDs, which is the convention we shall use. In order to keep the size of this type of control messages small, OPENET uses a short link/port ID of only 2 bytes.

Not all CPs along the path have to perform the same function, nor must the path be a simple one. Thus, for example, one can construct a path from, say,  $CP_1$  to  $CP_2$  and back to  $CP_1$  in a way that different functions (or none) are performed by the CPs in the forward and backward direction. For example, one can use such a message for the construction of a VC in which the initiator of the setup operation is not the source of the VC (third-party setup), an operation that proves useful for the construction of trees and for network management purposes. Another aspect of the source route traversal is the ability to trace back the path that the message took in the past. This function is very useful in handling error or blocking conditions. The reverse path is optionally accumulated by replacing the forward path indicators by the backward ones at each hop. If a message had traversed a route and reached a CP that determines an error condition, a new message can be constructed with the reverse route in which the previous action (e.g., bandwidth reservation) can be undone.

Another important service that can be provided with source route forwarding is datagram delivery between CPs, i.e., delivery of messages without a connection setup. Such a mechanism can be used for auxiliary services such as directory services.

The forwarding mechanism of the messages in this mode is based on the list of link IDs that appears in an address field at the head of the message. The IDs appear in the traversal order. A pointer variable which precedes this list, points to the particular link ID refers to at the current hop.<sup>2</sup> The pointer is advanced by the node to the next position before the message is forwarded. In many cases, it is also useful to construct the reverse path to be used, for example, by error messages that have to go in the exact opposite direction. Therefore, the source route forwarding mode has two operation options. The first, *reverse path accumulation*, specifies to each intermediate node to replace the already consumed link ID (outgoing from this node) by the ID of the link from which the message was received. The second, termed *direction*, instructs the node to follow the link list in a particular direction, hence, either incrementing or decrementing the pointer at each hop.

## 4.2 VC traversal forwarding mode

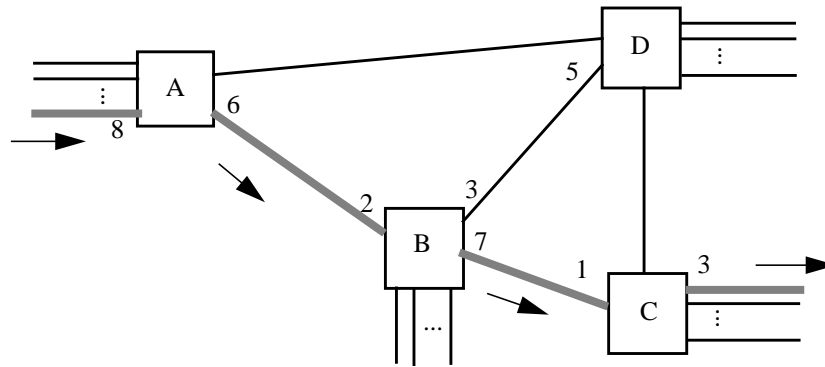
In the VC traversal forwarding mode, a message traverses a path from one CP to another according to the route of a previously established multi-hop VC. Thus, if a VC was established that traverses switches  $s_1, s_2, \dots, s_n$  a message using the VC traversal mode will be forwarded, along the control VCs, from the CP of switch  $s_1$  to the CP of switch  $s_2$ , and so on until it reaches the CP of switch  $s_n$ . (Note that when traversing a cluster in this mode the message passes through the CP only once and not once per switch). In other words, the control message is forwarded from a CP to a CP along the direction of an established VC<sup>3</sup> without using this VC for actual cell transmission.

A VC traversal message can traverse any type of VC, including multicast VCs. VC traversal can also be done in the reverse direction of the VC. This proves useful for recovery functions which arise during connection take down that results from topological changes or other error conditions such as lost messages. The CP keeps a version of the VC table(s) used by the switch hardware to switch cells from an input port to an output port (see Section 3.2). Hence, it is possible for CPs to exchange control messages along the same path as the established VC by examining the entries of the VC table.

### 4.2.1 VC traversal for unicast connections

To forward a VC traversal message along the path of a given unicast connection in a signaling channel, a CP needs to include the input VC label of the connection for the switch downstream along the path. To be more concrete, consider the following switch configuration in Figure 6. A unicast connection has been set up and the path is indicated by the shaded lines representing the actual links traversed by cells belonging to this connection.

- 
2. In general, the list of link IDs is sufficient to completely specify a path. However, OPENET provides mechanisms to include node IDs (22 byte ATM addresses) at arbitrary points in the source route for consistency check, for example.
  3. Note that the traversal actually follows the VC tables of the CP (not of the ATM switching subsystem) and hence might also traverse a *logical VC*, i.e., a VC that is not actually supported by the switch hardware.
-



**Figure 6.** Example switch configuration - unicast connections

Portions of the VC tables kept by the CPs of switches B and C are shown in Tables 4 and 5, respectively.

$P_{in}$	$VC_{in}$	$P_{out}$	$VC_{out}$
2	128	7	32
2	68	3	83
5	119	11	55
5	68	4	23
7	105	2	89

**Table 4.** Example of VC table at switch B - unicast connections.

$P_{in}$	$VC_{in}$	$P_{out}$	$VC_{out}$
1	32	3	50
4	22	6	38
3	91	1	105

**Table 5.** Example of VC table at switch C - unicast connections.

The shaded rows in the VC tables represent the connection in the direction shown by the arrows in Figure 6. In the VC traversal exchange mode, if the CP of switch B wishes to forward a control message to the next CP along the path of this connection, it creates a VC traversal message with a field containing the output label  $vc_{out}$  (32 in this case) of its own table corresponding to this connection. In cases where multiple links may connect two switches, it is also necessary for the VC traversal message to include a field containing either the output port num-

ber  $P_{out}$  (7 in this case) at switch B or the input port  $P_{in}$  at switch C (1 in this case).<sup>4</sup> (Note that each CP knows the port number at the neighbor switch associated with each of its local ports. Our architecture and the following description assumes that the output port is used in this field.) This message is forwarded to the CP of switch C via output port 7 in a separate VC that connects the CP of switch B to the CP of switch C. When the CP of switch C receives this message, it identifies output port 7 of the neighbor with input port 1 of switch C and looks at its own table for input port 1 and locates input label 32. From this entry, the CP of switch C determines that the message is to be forwarded via output port 3 with label 50. Thus, it creates a VC traversal message with an output label field 50 and output port field 3, and forwards it in the signaling channel of port 3. This process continues until the desired destination is reached.

The method of forwarding control messages from a CP to a CP along the path of an existing VC, described above, is referred to as the *Forward VC Traversal* or simply *VC Traversal*. The method of forwarding control messages from CP to CP in the reverse direction of an established VCI is referred to as the *Reverse VC Traversal*.

In the *reverse VC Traversal* mode, a CP forwards in the reverse direction of a given VC a message with fields containing the input label  $vc_{in}$  and the input port<sup>5</sup>  $P_{in}$ . The last row of each table above represents a VC in the opposite direction shown by the arrows in Figure 6. In our example above, the CP of switch B will receive a reverse VC traversal message with the label field 89 and input port field 6 at port 2. It identifies port 6 of switch A with output port 2 at switch B. In general, it will search through its table and attempt to locate the pair (2,89) in the ( $P_{out}$ ,  $VC_{out}$ ) columns. The CP of switch B then creates a reverse VC traversal message with the  $VC_{in}$  of this VC (105 in this example) and  $P_{in}$  (7 in this example) and forwards it to C via port 7. The CP of switch C, upon receiving this message from its input port 1, then searches its own table and locates the pair (1,105) from the output columns. From the table entry, it creates a reverse VC traversal message with the label field 91 and port field 3 and forwards it via output port 3, and so forth.

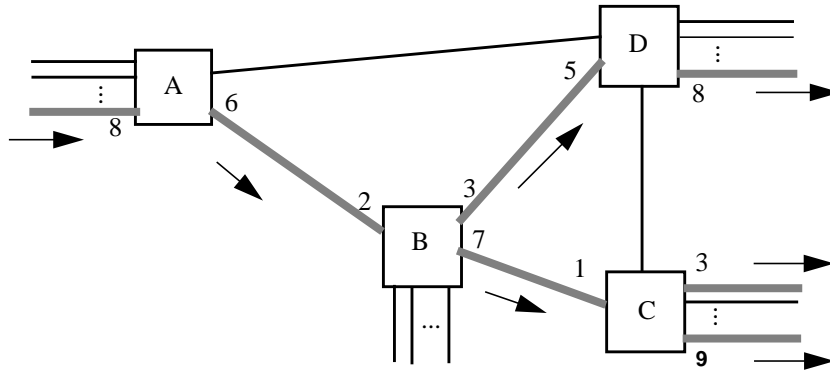
The reverse VC traversal method is generally less efficient than the forward VC traversal method since a search through an unsorted table may be necessary. Maintaining a duplicate table with sorted output entries solves this, but it nevertheless takes more storage and overhead. However, the reverse VC traversal method may be necessary to forward control messages along a unidirectional connection or a partially established bidirectional connection. The details of this are described later in Section 5.0.

#### 4.2.1.1 VC traversal for multicast connections

A VC traversal message can traverse the tree of a multicast connection in a way similar to traversing the path of a unicast connection. To forward a VC traversal message along the path of a multicast connection in a separate channel, a CP needs to include the *output VC label* and

- 
4. This information identifies the actual port the VC runs on in case there is more than one link between two switches and the traversal message is forwarded in a different link from the one used by the VC.
  5. The output port number of the neighbor switch of this link could be used instead. However, the same convention should be adopted by all CPs.
-

the *local output port number*<sup>6</sup> of the connection, as in the unicast case. The difference here is that each entry in the VC table of the CP may refer to more than one output port/label pair, depending on the structure of the tree. To be more concrete, consider the following switch configuration in Figure 7. A multicast connection has been set up (from root to leaves) and the path is indicated by the shaded lines representing the actual links traversed by cells belonging to this VC. The arrows indicate the direction of traversal of the message.



**Figure 7.** Example switch configuration-multicast connections

Selected entries in the VC tables for switches B, C and D are shown in Tables 6, 7, and 8, respectively.

$P_{in}$	$VC_{in}$	$P_{out}$	$VC_{out}$	$P_{out}$	$VC_{out}$
2	128	7	32	3	35
2	68	3	83	8	28
5	119	11	55	4	42
3	1068	2	1023	-	-
7	1105	2	1023	-	-

**Table 6.** Example of VC table at switch B-multicast connections.

$P_{in}$	$VC_{in}$	$P_{out}$	$VC_{out}$	$P_{out}$	$VC_{out}$
1	32	3	50	9	77
4	22	6	38	7	54

**Table 7.** Example of VC table at switch C-multicast connections.

6. The port information is needed in case more than one link may connect a switch pair. As before, the input port of the neighbor switch could be used instead.

$P_{in}$	$VC_{in}$	$P_{out}$	$VC_{out}$	$P_{out}$	$VC_{out}$
3	1091	1	1105	-	-
9	1128	1	1105	-	-

**Table 7.** Example of VC table at switch C-multicast connections.

$P_{in}$	$VC_{in}$	$P_{out}$	$VC_{out}$
5	35	8	50
6	53	3	36
8	1018	5	1068

**Table 8.** Example of VC table at switch D-multicast connections.

The shaded rows in the VC tables represent the multicast connection in the direction shown by the arrows in Figure 7.

In the *Multicast Forward VC Traversal*, a CP forwards a control message to all downstream CPs by locating an entry in its VC table matching the label in the VC traversal message it receives from a previous CP. For example, supposing that the CP of switch B receives a VC traversal message at port 2 with label 128 and port 6; it will create two messages with different label and port fields. The first message will be sent via port 7 with label and port fields 32 and 7, respectively. Similarly, the second message will be sent via port 3 with label and port fields 35 and 3, respectively. The CP at switch C receives the message at port 1 and looks up its own VC table. It also forwards two messages, one via port 3 with label 50 and port 3, and the other via port 9 with label 77 and port 9. On the other hand, the CP of switch D receives B's message at port 5 and forwards this message via port 8 with label 50 and port 8. This process continues until the message arrives at a leaf node.

It is also possible to use the forward VC traversal mode to send control messages from a leaf node back to the root for an established *source tree*. Basically, a source tree consists of a number of source nodes (the leaves) and a single destination node (the root) where cells are directed from the leaves to the root. The forwarding of a message along a source tree from a single leaf to the root using forward VC traversal is identical to the unicast case and will not be detailed here.

In *Multicast Reverse VC Traversal*, the CP forwards messages in the reverse direction of either a multicast tree or a source tree. As an example, consider the reverse traversal of a source tree. The rows in the VC tables below the double line represent parts of a *source tree*. Control messages are to be forwarded in the direction shown by the arrows in Figure 7, which is the reverse direction of the source tree. In this case, the CP of switch B will receive a reverse VC traversal message from A with fields containing the label 1023 and port 6 at port 2. It will then search through its VC table locating *all* occurrences of the pair (2,1023) in the output fields. There are

two rows in the table matching this search and thus it forwards two messages, one to the CP of switch C with the label field 1068 and port field 3 via port 3, and one to the CP of switch D with the label field 1105 and port field 7 via port 7. The CP of switch C in turn locates two entries of the pair (1,1105) in its own table and forwards two messages, one with label 1091 and port 3 via port 3 and one with label 1128 and port 9 via port 9. The CP of switch D also receives the message from B and locates a single entry (5,1068) in its table and forwards a message with label 1018 and port 8 via port 8. The process continues again until the message arrives at a leaf node. The reverse VC traversal of a multicast tree is similar to the reverse traversal of a unicast connection and will not be detailed here.

For the reverse VC traversal of a source tree, the need to search for multiple occurrences of output port/label pairs in the table implies that the entire table must be traversed to determine if all occurrences have been found. In this situation, maintaining a duplicate table with entries sorted by output port/labels may be justified.

The forward and reverse VC traversal methods described above relies on the existence of a *uni-directional* connection only. If a bidirectional unicast connection is available, then the message forwarding along any direction can be done in either the forward or reverse traversal mode. A multicast connection can be unidirectional or bidirectional, in general. If a bidirectional multicast connection exists, then the VC traversal can also be performed in either forward or reverse directions.<sup>7</sup> The forward VC traversal mode should be used in general since it is more efficient. However, the reverse traversal may be the only possible way to traverse a connection under some exception conditions, such as a partially set up or partially released connection.

#### **4.2.1.2 Handling of transient conditions under VC traversal**

With proper design, VC traversal can be used on connections that are not complete, as in some transient conditions such as partially established or partially taken down connections. Since the ATM forum UNI 4.0 specifies bidirectional unicast and unidirectional multicast connections, we describe the handling of transient conditions for both of these types of connections here. The VC traversal mode of forwarding control messages to CPs along the path of a given unicast connection or the tree of a multicast connection works well on any established VCs, where all the labels are in place for either direction (*logical* labels for the reverse direction are required in the case of a unidirectional multicast connection). However, it is possible to use the VC traversal forwarding mode even when a VC has not been completely established or when the VC has been partially taken down.

To support error recovery during connection establishment, connection maintenance, and connection take down, a combination of forward and reverse VC traversal may be used. In general, the forward VC traversal method should be used if labels are available for forward traversal. The reverse VC traversal method should be used if it is not possible to use the forward traversal. A CP receiving a reverse VC traversal message should try to determine if it is possible to continue with a forward VC traversal mode, and if so, should use the forward VC traversal

---

7. In the case of a unidirectional multicast connection, it is possible to maintain a set of reverse direction logical labels (a logical source tree). See Section 3.2.

mode. A CP receiving a forward VC traversal message should always continue with a forward VC traversal message. Note that the transient condition where a CP receiving a forward VC traversal message is able to use only reverse traversal but not forward traversal cannot exist.

#### **4.2.1.3 Handling of exception conditions under VC traversal**

If a CP receives a VC traversal message but is unable to locate the appropriate VC label and port in its VC table, an exception condition is said to have occurred. There may be error scenarios caused by the uncompleted release of VCs which lead to exception conditions in the use of VC traversal. Under exception conditions, it is not possible to proceed any further with either the forward or the reverse VC traversal. The CP should abort the forwarding of messages and initiate a release procedure in the opposite direction of traversal to relinquish the resources associated with the partially released VC involved.

### **4.3 Control packets**

To achieve the control functions, the CPs need to communicate with one another based on the Source Route Forwarding Mode (SRFM) and the VC Traversal Forwarding Mode (VCTFM). We define the general packets for these two forwarding modes as follows (details on the formats and contents of each of these packets appear in [17]):

**Source route packet:** Source route packet is used to support the source route forwarding mode. In the source route forwarding mode, a message traverses a path from CP to CP according to a route that is explicitly contained within the message. Each CP along the path not only forwards the message to the next CP along the path but also performs a certain function that is also indicated in the message. The N\_SETUP message uses the source route traversal forwarding mode.

**VC traversal packet:** A VC traversal packet is used to support the VC traversal forwarding mode. In the VC traversal forwarding mode, a message traverses a path from one CP to another according to the route of a previously established VC. The VC traversal packet contains the standard PNNI packet header, a port ID, and a VC/VP traversal label field. All VC traversal packets also contain a return label to be used when the packet reaches a CP where the indicated traversal label does not exist. In this case, the return label is required for cleaning up the connection in the reverse direction. The N\_SETUP\_ACK packet, the N\_CONNECT packet, and the N\_CONNECT\_ACK packet are of the forward VC traversal type.

#### **4.3.1 Consistency check**

In general, each CP receiving a VC traversal message must check for consistency of the label indicated in the message. If a VC traversal message arrives at a CP in which the VC label entry is not found in the table, a VC release procedure (see Section 5.2) must be initiated by the CP, typically to the opposite direction that the VC traversal message came from. The exact procedure is described in the corresponding sections.

## **5.0 Connection Control and Maintenance**

---

The main tasks associated with connection control are connection setup, connection take down and connection maintenance. All these tasks should be performed in an efficient and reliable manner. The connection setup we employ is source-based and includes the update of routing tables in intermediate switches and the bandwidth reservation along the chosen routes. The connection can be either a unicast, namely a connection between the source and a single destination; a single source multicast, namely a connection between one source and several destinations; or a multi-source multicast, namely a connection between several sources and several destinations. Connection take down is used for bandwidth release upon the termination of calls, or upon failures that disconnect the call. Connection maintenance is needed to insure fault-tolerant operation of the network and the release of unused reserved bandwidth if normal take down is unsuccessful due to failures.

We will first describe the connection setup operation from one source to a single destination (unicast), assuming no errors and no failures along the connection. We will then describe the same operation for a multicast connection. Errors and failures will be addressed when connection maintenance is addressed. Note that any multicast connection setup can serve also as a unicast connection setup. As will become apparent from the following description, using multicast for a unicast connection setup is wasteful. A unicast connection setup is an operation that is expected to be very frequent, and therefore it is crucial to devise it to be as efficient as possible. The unicast connection setup proposed in the following section is optimized (with respect to the forwarding modes that are used), and it is by far more efficient than the multicast connection setup described in Section 5.4.

As described in Section 3.2, each CP has a swap table (*VC table*) for each of its switches. Each entry in the table corresponds to a connection (*VC*) that traverses the corresponding switch. An entry in the table is created whenever a new *VC* through the switch is attempted to be set up, and is deleted from the table upon the completion of the connection or upon unsuccessful setup. The amount of bandwidth reserved for the connection is also available in a resource usage table maintained by the CP.

### **5.1 Unicast connection setup**

In the following, we give a general description of the unicast connection setup process in OPENET. The detailed signaling and message formats are given in Section 5.1.1. A unicast connection setup is initiated by a user (host) when it needs to communicate with another user (host) in the network. (In Section 5.9, we will describe an operation of establishing a unicast connection between two hosts, by a third host, say a manager.) The host sends to its CP a SETUP message (defined by UNI) that contains its address, the destination address, the class of service it requires, and other parameters as per the specification of the UNI. The CP maps the request into one of the offered classes of service and a representation of the traffic intensity. The traffic intensity will be represented by two values: MAX and MIN, indicating a region of

acceptable operation. One of the reasons for such a setting is allowing the user to make more flexible decisions based on network state. Another reason is that the mapping of user defined service to network service classes leaves some tolerance for traffic specification. Note that while bandwidth negotiation is not part of current UNI, OPENET supports it anticipating future UNI upgrades.

The CP then computes the best (loop-free) route for the corresponding class of service based on its local information on links utilization. The route must ensure that the available bandwidths in all its links fulfill the user's request.

The CP then constructs an N\_SETUP message (defined by OPENET) that uses the source route forwarding mode. The message must contain, in addition to its type designation and route, the MAX and MIN values for the bandwidth reservations, the function to be performed by every link and *label* fields that are changed by the corresponding CPs along the route. It also carries the relevant end-to-end parameters that were part of the user UNI SETUP.

The N\_SETUP message is then forwarded *downstream* from the source CP to the destination CP. Each CP that receives the N\_SETUP message checks whether the corresponding switch can accommodate the connection. If the available bandwidth for the requested class is greater than the current MAX value, a quantity MAX of the bandwidth is reserved (by reserved we mean that until this portion of the bandwidth is released, it cannot be used by any other connection), and the CP forwards the N\_SETUP message to the next CP on the route. If the available bandwidth is between the MIN and MAX values, say M, then a quantity M of the bandwidth is reserved, the MAX value in the N\_SETUP message is replaced by M and the CP forwards the updated N\_SETUP message. In each of these cases, the CP updates the *backward label* field before forwarding the N\_SETUP message to the label it chose for the connection. In addition, the resource usage record of this VC in the VC-table is updated accordingly. If the available bandwidth is smaller than the MIN value, then a connection cannot be established and the connection setup process is aborted by the CP creating an N\_RELEASE message that is forwarded *upstream* from the current CP back to the source CP.

The N\_RELEASE message, which is of the VC traversal type, contains in addition to the label field the reason of the failure (insufficient resources, failed link, failed CP, unavailable host, etc.). The label field is generic to the VC traversal messages and is used to forward the message backward to the source CP along the route that has already been set up, and is updated by each CP along the reverse path. Upon receiving an N\_RELEASE message, each CP along the reverse path releases the bandwidth it already reserved (so this bandwidth becomes available to other connections), and forwards the N\_RELEASE message upstream. In addition, the corresponding entry is deleted from the VC tables. When the message arrives at the source CP, it informs the user of the failure to establish a connection.

When an N\_SETUP message arrives at the destination CP, it forwards a UNI SETUP to the destination host and generates a N\_SETUP\_ACK message that is forwarded upstream to the source CP along the same route of the established connection. The N\_SETUP\_ACK message, which is of the forward VC traversal type, contains the label field and the MAX value con-

tained in the N\_SETUP message that arrived at the destination CP. In addition, the N\_SETUP\_ACK message contains an additional label field in which the CP informs the previous CP along the route which label to use for the  $vc_o$  for that connection.

Upon receiving an N\_SETUP\_ACK message, each CP along the reverse path updates the bandwidth reserved for the connection to that indicated by the message (its original reservation was at least that value), updates the label field ( $ve, c_o$ ), which now becomes permanent for this connection, and sends the message upstream. When the message arrives at the source CP, the connection is set up, and the source CP lets the source host start its communication with the destination host using the current MAX parameter. When the destination host responds with a UNI CONNECT to the destination CP, the CP sends N-CONNECT upstream to the source CP using the forward VC traversal mode. When the N\_CONNECT reaches the source CP, it is converted back to a UNI CONNECT and delivers to the source host. In parallel, the message N\_CONNECT\_ACK is sent back to the destination host using the forward VC traversal mode for error recovery purposes. Both N\_CONNECT and N\_CONNECT\_ACK do not impose any additional processing at the intermediate nodes (except for the VC traversal forwarding), and are used for the task of supporting the UNI signaling exchange. When the N\_CONNECT\_ACK reaches the destination CP, it delivers a UNI CONNECT ACKNOWLEDGE message to the destination host. The setup procedure is now completed.

### 5.1.1 OPENET and UNI signaling for unicast connection setup

In this section, we define the set of control messages carried along the CPs for the purpose of managing the various connections used by the network users. We use the terms *source CP* (S\_CP) and *destination CP* (D\_CP) to refer to CPs that reside on the network side of the UNI of the call originator and call destination, respectively. Also, the term *intermediate CP* (I\_CP) refers to the OPENET network CPs that perform processing and forwarding of OPENET control messages within the ATM network. The calling party and the called party refer to the call originating entity and the call destination entity on the user sides of the UNI, respectively. The called and calling parties exchange messages using the UNI format with their respective network control entities (there is no direct signaling between the two parties).

The call setup procedure for OPENET is described by a sequence of messages exchanged among the S\_CP, I\_CP's, and D\_CP, which is triggered by the calling party (caller) on the user side UNI. The caller sends a "SETUP" message (defined by UNI 4.0, see [1]) to the CP on the network side of the UNI (S\_CP). This message must include the ATM user cell rate, QoS parameter, and called party number, etc., and optionally include AAL parameters, calling party number, etc. Notice that the ATM user cell rate is specified for both forward and backward directions in this SETUP message, following the UNI 4.0 specification that all point-to-point connections are bidirectional (and might involve different bandwidth requirements in both directions). The S\_CP, upon receipt of the SETUP message, potentially performs a translation of the ATM address to a network address. After the transmission of the SETUP message, the calling party changes from call state "Null" (U0) to "Call Initiated" (U1) and starts timer "T303." The default timer value is 4 seconds. This timer is used to support retransmission of SETUP messages by the caller. If the S\_CP responds with a "CALL PROCEEDING" message (see below), the user changes to state "Outgoing Call Proceeding" (U3) and starts timer T310

with a default value of 10 seconds. This timer is used to trigger “RELEASE” messages if no response (via CONNECT messages) from the called party is received.

When the S\_CP determines locally that access to the requested service is authorized and available, it may send a “CALL PROCEEDING” message to the user to acknowledge the SETUP message and to indicate that the call is being processed. The call state on the network side (S\_CP) transits from “Null” (N0) to “Call Initiated” (N1) and then to “Outgoing Call Proceeding” (N3). Based on the information in the user SETUP message, the S\_CP maps the request into one of the offered classes of service and representation of the traffic intensity. It then computes route for the corresponding class of service. While the S\_CP is in state N1 or N3, it will create an internal network control message (which we call “N\_SETUP” to distinguish it from the UNI SETUP message). This N\_SETUP message uses the source route traversal forwarding mode, and contains information to route and reserve bandwidth (at both directions) along a pre-determined path. The S\_CP starts and maintains a separate timer (termed “Setup Timer”) upon whose expiration it will retransmit the N\_SETUP message. This timer will be cleared upon receipt of the N\_SETUP\_ACK message described below. The N\_SETUP message traverses the CPs along the route specified by the S\_CP. This message also performs at each hop, reservation (for both directions) and VCI label exchange (for the backward direction).

The OPENET architecture supports negotiation of resources such as bandwidth, although the current UNI does not provide the users with such a capability. To support bandwidth negotiation, the S\_CP includes two values in its message to the next CP containing the minimum acceptable bandwidth,  $BW_{min}$ , and the maximum desired bandwidth,  $BW_{max}$ . At each I\_CP, the available bandwidth,  $BW_{avail}$ , is compared against these two values. If  $BW_{avail}$  is larger than or equal to  $BW_{max}$ , then these same maximum and minimum values are passed on to the next CP along the path. On the other hand, if  $BW_{min} \leq BW_{avail} < BW_{max}$ , then  $BW_{max}$  is set to  $BW_{avail}$  in the N\_SETUP message to be forwarded to the next CP. Finally, if  $BW_{avail} < BW_{min}$ , then the N\_SETUP fails and a release procedure will be initiated to release previously-allocated resources for this connection. At the D\_CP, the value  $BW_{max}$ , after being adjusted according to the same procedure as described before, will contain the maximum bandwidth for this connection that can be provided by *all* the switches along the path. At this time, assuming that the bandwidth reservation is successful, i.e., all bandwidths are available in both directions, the D\_CP responds to the S\_CP by sending back a “N\_SETUP\_ACK” message using the VC traversal forwarding mode. The purpose of this message is to first acknowledge the source about the successful arrival of the N\_SETUP to the D\_CP and to adjust the reserved bandwidth parameters to the maximum value that was available at all switches along the path. This message allows the S\_CP to recover from loss of control messages in a timely manner before the user timer expires.

The S\_CP stops its Setup Timer when the N\_SETUP\_ACK message is received. Meanwhile, the D\_CP informs the called party of the connection attempt by sending a SETUP message to the user side UNI and starts timer T303 and enters the “Call Present” (N6) state. This SETUP message must include the connection identifier, as specified by the UNI. The called party transits from state “Null” (U0) to “Call Present” (U6) after it received the SETUP message. The SETUP message contains all the information required by the called user to process the call.

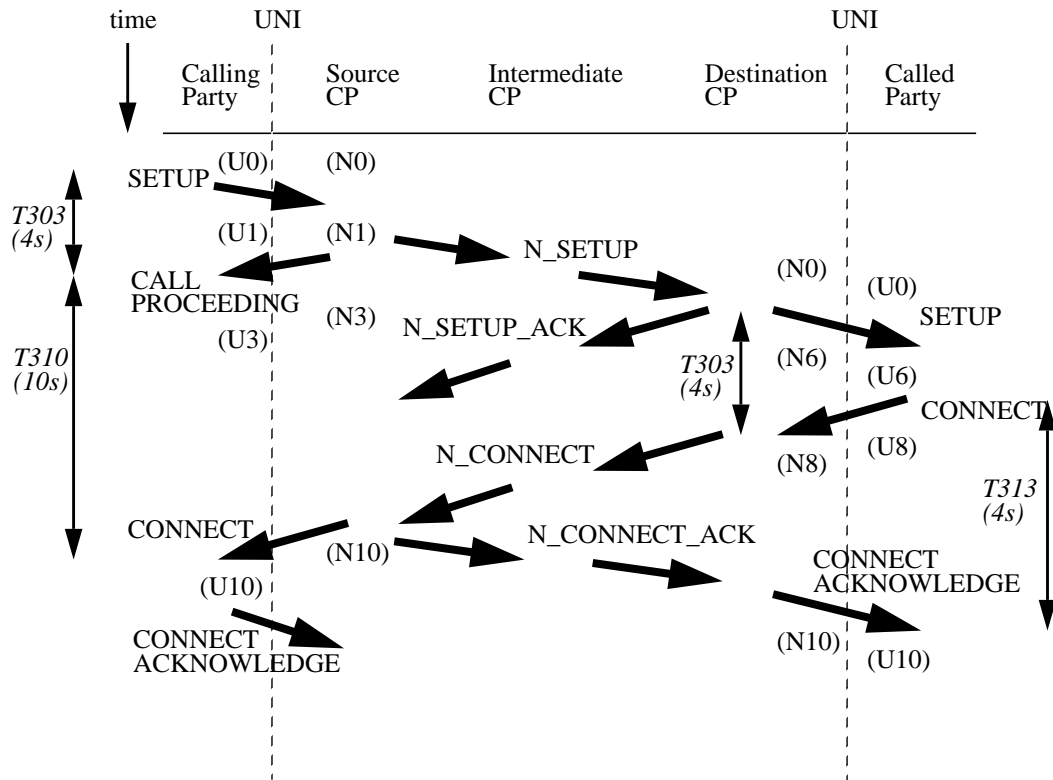
Upon receipt of a SETUP message by the called party, if the user wishes to accept the incoming call/connection, the user responds with a CONNECT message and enters the “Connect Request” (U8) state. (A user may wish to respond first with a “CALL PROCEEDING” message and enter the “Incoming Call Proceeding” (U9) state if it cannot respond with a CONNECT message before the timer T303 expires.) Upon sending the CONNECT message, the user starts timer T313. If a CONNECT ACKNOWLEDGE message has not been received before the timer expires, the user sends a RELEASE message. The D\_CP, upon receipt of the CONNECT message, will initiate procedures to send an “N\_CONNECT” message towards the calling party. The D\_CP starts and maintains another timer (called the “Connect Timer”) in order to recover from the loss of the N\_CONNECT message or its acknowledgment inside the network. If the “Connect Timer” expires before the N\_CONNECT\_ACK message is received, the D\_CP retransmits the N\_CONNECT message. The N\_CONNECT message serves to carry optional AAL parameters present in the CONNECT message sent by the called party. In addition, the N\_CONNECT message also performs the same functions as the N\_SETUP\_ACK message. This allows the N\_CONNECT message to be used as a retransmission for lost N\_SETUP\_ACK messages. When the N\_CONNECT message arrives at the S\_CP, the connection is set up, and the S\_CP lets the calling party start its communication by sending a CONNECT message across the UNI. The S\_CP enters the state “Active” (N10). It also sends a “N\_CONNECT\_ACK” message back to the D\_CP to indicate that the N\_CONNECT message has been successfully received. On receipt of the CONNECT message, the calling party stops timer T303 or T310, sends a CONNECT ACKNOWLEDGE message and enters the state “Active” (U10). The S\_CP does not take any action on the receipt of the CONNECT ACKNOWLEDGE message when it perceives the call to be in the Active state. At this point an end-to-end connection is established.

Finally, on receipt of the N\_CONNECT\_ACK message at the D\_CP, it will send a CONNECT ACKNOWLEDGE message to the called party. The D\_CP transits to state “Active” (N10). This state indicates that the D\_CP has awarded the call to the called user. The CONNECT ACKNOWLEDGE message indicates completion of the ATM connection for the D\_CP/callee interface. From the UNI point of view, when the called party receives a CONNECT ACKNOWLEDGE message, there is no guarantee of an end-to-end connection until a connect message is received at the calling user. However, with this exchange, a CONNECT message is guaranteed to have been received by the calling party when the called party receives the CONNECT ACKNOWLEDGE.

The illustration of this exchange is depicted in Figure 8.

## **5.2 Unicast connection clearing**

In normal operation of the network, clearing (take down) operations are needed to release reserved bandwidth upon the termination of connections. In addition, take down operations are needed to release reserved bandwidth when failures cause the disconnection of the source from its destinations or a degradation of the quality of service that was previously guaranteed.



**Figure 8.** UNI signaling and OPENET network control for unicast connection setup

### 5.2.1 End of connections

The release of reserved bandwidth when a connection ends is simple. When the source user no longer needs a specific connection it sends a **RELEASE** message (defined by UNI 4.0) to its CP. This source CP then forwards an **N\_RELEASE** message (defined by OPENET) along the setup route according to the VC traversal mode. Each CP that receives the message releases the corresponding bandwidth at the appropriate switch, erases the corresponding entry from the VC table, and forwards the message. In the other direction, when the destination user no longer needs a specific connection, it sends a **RELEASE** message to its CP. This destination CP forwards that message along the setup route according to an appropriate<sup>8</sup> VC traversal mode, and each CP does the same operations as above.

8. Forward or reverse depending on whether the connection is bidirectional or unidirectional.

### **5.2.2 Failures**

Different types of failures may occur in the network that will disconnect the source from its destinations. Among them are link failures, switch failures, and user/host failures. A CP that detects a failure is responsible for starting the process of releasing the reserved bandwidths of all connections that are affected by the failure. Therefore, when a CP detects a failure, it first determines the connections affected by the failure by checking its VC table. Then, for each affected connection, it sends an N\_RELEASE message to the CPs in both the upstream and downstream directions along the established VC using the VC traversal mode. Each CP that receives the message releases the corresponding bandwidth at the appropriate switch, erases the corresponding line from the VC table, and forwards the message. If the message arrives at a CP in which the corresponding line in the VC table has already been erased, it does not forward it.

### **5.2.3 OPENET signaling for unicast connection clearing**

The clearing procedure for OPENET can be initiated by either the user or by the network. In the former case, the user initiates clearing by sending a “RELEASE” message across the UNI to the S\_CP, transits to “Release Request” (U11) state, and starts timer T308 with a default value of 30 seconds. The S\_CP, upon receipt of the RELEASE message, enters the “Release Request” state (N11) and initiates the procedure for clearing the network connection to the remote user. This is accomplished by sending an “N\_RELEASE” message across the ATM network. Upon receipt of the N\_RELEASE message, the D\_CP transmits a RELEASE message across the UNI to the called party, transits to “Release Indication” (N12) state and starts timer T308 with a default value of 30 seconds. The called party will reply with a “RELEASE COMPLETE” message and return to the Null state (U0). On receipt of the RELEASE COMPLETE message, the D\_CP stops timer T308, releases both the virtual channel and call reference, and returns to the Null state (N0). In the case of a UNI RESTART message, the S\_CP will clear all connections as if a RELEASE message was received individually for each connection.

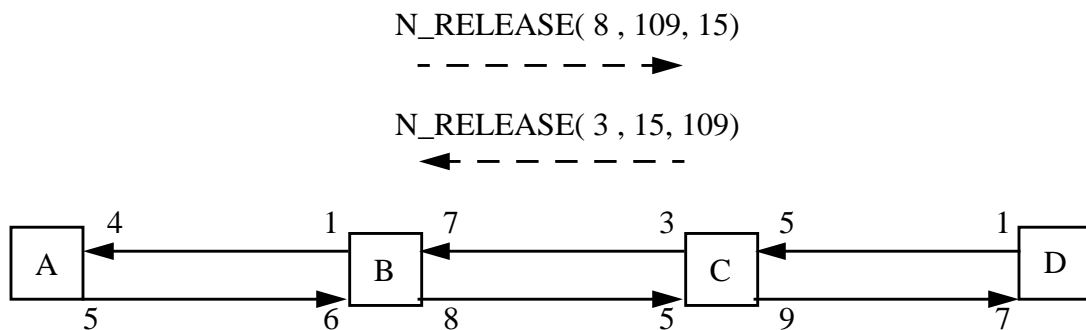
In the following subsections, we describe the protocol for the return of VC labels used in the VC tables at each of the CPs. The reserved bandwidth for a given connection can also be released at the same time as the VC labels are being released. However, in order to return the bandwidth to the network as soon as possible, it may be more efficient to relinquish the bandwidth for the VC using the N\_CHANGE message protocol (described later), which is a reliable end-to-end protocol, before sending the N\_RELEASE.

### **5.2.4 One pass release**

In this protocol, we make the optimistic assumption that errors occur very infrequently. Therefore, only a single N\_RELEASE message will be transmitted from the CP initiating the clearing procedure (S\_CP or D\_CP). As the N\_RELEASE message is forwarded from the S\_CP to the I\_CP and then to the D\_CP (or vice versa), the VC labels associated with the connection will be removed from the VC table at the corresponding CP. The normal clearing procedure completes when the N\_RELEASE message completes the path.

**5.2.4.1 Clear collision**

Clear collision occurs when N\_RELEASE messages are sent from opposite directions in the connection. This happens when, say, both the calling party and the called party send RELEASE messages across the UNI. When the N\_RELEASE messages cross each other, they arrive at CPs where the labels have already been removed from the VC table. When an N\_RELEASE message arrives at a CP where the labels indicated in the message do not exist or are inconsistent with the VC table at the CP, the N\_RELEASE message will be discarded and ignored. This case is termed (release inconsistency) in the forward direction, as the inconsistency is identified by observing that the forward label contained in the N\_RELEASE message doesn't exist in the VC table. This case is described in Figure 9 where we use the convention N\_RELEASE(Port ID, VC/VP forward label, VC/VP reverse label).



**Figure 9.** Release consistency in the forward direction.

**5.2.4.2 Lost messages**

In the unlikely event that the N\_RELEASE message is lost, the VC labels that have not been returned will be recovered using the refresh mechanism.

**5.2.5 Consistency check**

In general, each CP receiving a VC traversal message must check for consistency of the label indicated in the message. If a VC traversal message arrives at a CP in which the VC label entry is not found in the table, an N\_RELEASE message must be sent by the CP in the opposite direction that the VC traversal message came from. In order to do this, every VC traversal message must also include the reverse traversal label (the return label) so that N\_RELEASE message can be sent in the reverse direction when an inconsistency is encountered. The only exception to this is the N\_RELEASE message itself, which will simply be ignored when an inconsistency is found (see Figure 10).

**5.2.6 Label reuse**

When a label is removed from the VC table of a CP, it is possible that the label is reused immediately under some situations. If an N\_RELEASE message for an old and partially cleared connection now arrives at the CP, there is a potential danger that this message may tear down the new connection. This problem is handled with an extra checking step during the SETUP phase. The extra checking determines if the newly received neighbor's label is not used by any other

(old) connection. If this is the case, the old connection is released. An example of this case is described in Figure 10. We use the convention  $N\_SETUP(\text{next Port ID, VC/VP Backward Label, VC/VP Forward Label})$ . In Figure 10, when node B receives  $N\_RELEASE(5, 80, 2)$  it releases label 15 in the VC table of input link 7, and sends  $N\_RELEASE$  message to node C. This  $N\_RELEASE$  message is lost, and hence part of the connection from node C to node D is not released.  $N\_SETUP$  message destined to port 8 arrives at node B, which allocates the released label 15 to the corresponding connection and sends  $N\_SETUP(9, 15, -1)$  to node C. When this  $N\_SETUP$  message arrives at node C, it checks if the backward label 15 used on output port 3 appears in any of its input ports tables. Since it appears in the VC table of input link 5, it recognizes that an unreleased old segment of a connection exists, releases the bandwidth and labels for that connection and sends an  $N\_RELEASE$  message for that segment before sending the  $N\_SETUP$  message. The different events are marked with increasing numbers where larger numbers indicate later times.

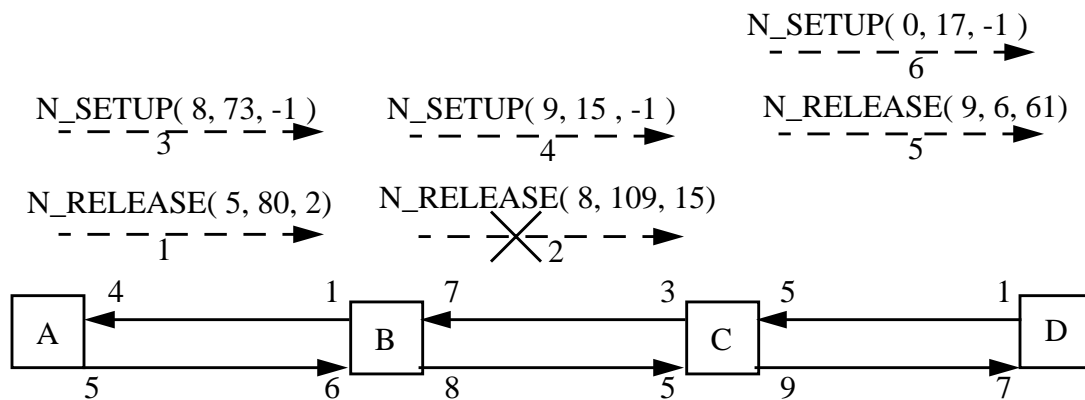


Figure 10. Setup consistency in the backward direction

Another situation in which a consistency check is needed to prevent an  $N\_RELEASE$  message of an old connection from releasing a newly established connection is described in Figure 11. When node B receives  $N\_SETUP(1, 21, -1)$ , it allocates label 15 in the forward direction. When node B receives  $N\_RELEASE(3, 15, 109)$ , it discovers that label 15 is associated with label 21 of the new connection and not label 109 that appears in the  $N\_RELEASE$  message. This inconsistency causes node B to ignore this message.

The  $N\_RELEASE$  packet is of the VC traversal type. It can be forwarded in either direction.

### 5.3 Unicast connection maintenance

Connection maintenance is important for “cleaning” the network from malfunctioning that is associated with loss of control messages, wrongly addressed control messages, table errors, and any other failures that are not detected directly by the CPs. These events are expected to be rather rare, and therefore, the connection maintenance we use is based on the source CPs sending periodic  $N\_REFRESH$  messages for each established connection at a low rate. An  $N\_REFRESH$  message contains the VC labels and port ID and connection parameters and is

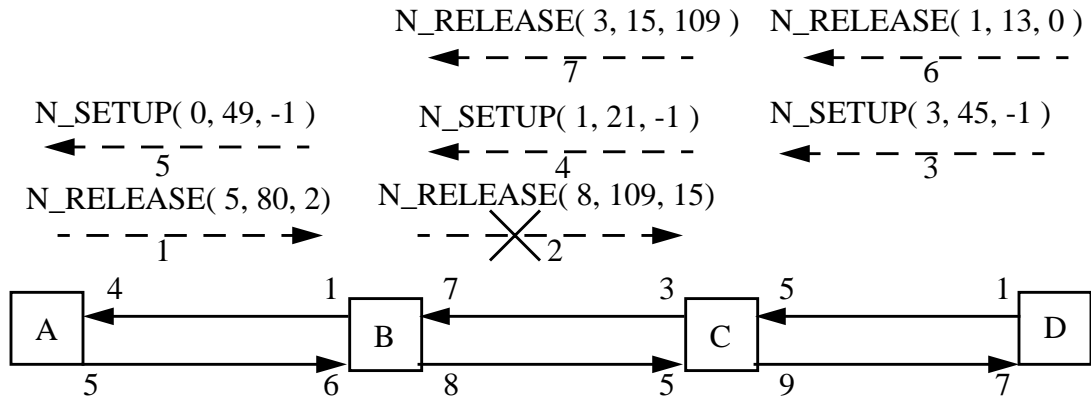


Figure 11. Release consistency in the backward direction

forwarded along a setup path according to the VC traversal mode. These N\_REFRESH messages allow the intermediate CPs along the connection to track the connection existence and its parameters. For the source CP to know a connection still exists, the N\_REFRESH messages have to be acknowledged by the destination CPs. The acknowledged N\_REFRESH messages are forwarded in the reverse direction according to the appropriate VC traversal mode. For even better assurance, we can let the destination CP send N\_REFRESH messages periodically using the (appropriate) VC traversal mode.

Each CP will keep for each of the established connections a parameter that indicates how often N\_REFRESH messages are expected for that connection. If the number of N\_REFRESH messages (for a particular connection) that pass the CP within a specific period is less than this parameter, the CP will conclude that the connection has been disrupted, and will start a disconnect procedure as explained in Section 5.2. This parameter is not expected to be uniform for all connections. Connections that hold large bandwidth portions will require more frequent N\_REFRESH messages than “light” connections. To illustrate the design trade-off, consider the following design example. Assume that the target rate of control messages sent by a CP should not exceed 2000 messages/sec, and assume that our design point is to spend at most 10% of these messages as refresh messages, i.e., 200 refresh messages/sec. Assume that this CP has at most 100 ports in its cluster, each has at most 1000 established connections. If the number of refresh messages is evenly allocated between the ports, the CP sends two refresh message/sec on each port. We allocate this budget in the following way. For each connection with less than 0.5% of the port capacity, the CP will send a refresh message every 1000 seconds. For each connection with more than 0.5%, 5%, and 50% of the port capacity the CP will send a refresh message every 200, 20, and 2 seconds, respectively. Note that the rate of the refresh period is negotiable in the sense that the N\_SETUP message contains the source CP intended refresh period. This period can be increased by the intermediate nodes and adjusted via the returning N\_SETUP\_ACK message.

OPENET supports the changing of reserved bandwidth in a connection. The N\_CHANGE message is used to decrease the reserved bandwidth along a path when it is desired by the

source user. It is always possible to lower a bandwidth requirement, but the increase in bandwidth is on an availability basis. The N\_CHANGE message can be used in combination with the N\_RELEASE message to provide a highly reliable mechanism for the return of critical resources used by large bandwidth connections.

The N\_CHANGE message contains VC labels identifying the connection, the minimum and maximum bandwidth desired for this connection, and a “refresh period” (as in the N\_REFRESH message). The N\_CHANGE message is acknowledged by the N\_CHANGE\_ACK message, which contains the final negotiated value. The N\_CHANGE, N\_CHANGE\_ACK, and the N\_REFRESH packets are of the forward VC traversal type.

## **5.4 Multicast connection setup**

In this section, we discuss only a single source multicast connection setup. Clearly, a multi-source multicast connection can be achieved by each source initiating a single source multicast connection setup. Although this is not an efficient solution, it seems to be the only solution that is supported by the currently available user interface. Better solutions can be devised using distribution trees, but they will require changes in the user interface.

A multicast connection setup is initiated by a user when it needs to communicate with several other users in the network. The user sends its CP a request that contains the destination addresses and the class of service it requires. As in the unicast case, the CP maps this to a network class of service along with a MIN-MAX representation of traffic intensity. The CP then computes low cost routes for the corresponding class of service based on its local information of links states. These routes form a tree rooted at the source host and each of its leaves is one of the destination hosts. As with the unicast connection, the routes must ensure that the available bandwidths along the links of the tree guarantee the user’s request.

OPENET supports and extends the setup of multicast connections defined by UNI 4.0 by providing two ways to create a multicast VC. The first method, called *Complete Setup*, is based on a single setup message that traverses the whole tree and creates a multicast tree in a single round. The second method, called *Step-by-Step Setup*, is based on several setup messages that traverse different parts of the tree (that may partially overlap). The step-by-step method allows the root to add destinations to the multicast tree one by one (as in UNI 4.0), yet is more general in the sense that more than one destination can be added, provided the path used by the setup message is a simple one (see below). The two approaches to multicast connection setup are described in the following subsections.

### **5.4.1 Complete setup - A single message multicast setup**

Based on the best routes it determined (and the tree that resulted), the source CP computes a traversal of the tree along a *connected path*. This path starts and ends at the source CP, and traverses each link of the tree exactly twice. The setup process is achieved by an N\_SETUP message (which is of the source traversal route type) indicating *point-to-multipoint* connection, and contains the source route, the function to be performed by every link, the label field, and

the MIN and MAX values, much like the regular N\_SETUP message. The possible functions used are: RSRV\_FWD (RF), RSRV\_BWD (RB), RSRV\_BOTH (R2), EXCHANGE (XC), NULL (NU), END\_POINT (EP), and NODE\_ID (ND).

A CP that receives the N\_SETUP message with the NULL function just forwards the message to the next CP according to the route contained in the message. A CP that receives the N\_SETUP message with one of the three reserve functions (RF, RB, R2) checks (as in the unicast case) if the requested service can be supported. If the resources cannot be secured (or, in the case of a destination, the user is unavailable), an N\_RELEASE message is sent back towards the source CP (see below). If the residual bandwidth (for the class of service required by the connection) is greater than the current MAX value, a quantity MAX of the bandwidth is reserved and the CP forwards the N\_SETUP message to the next CP in the route. If the residual bandwidth is between the MIN and the MAX values, say M, then a quantity M of the bandwidth is reserved, the MAX value is updated to M and the CP forwards the updated N\_SETUP message. In each of these cases, if the Forward Label field in the previous N\_SETUP message is the null label, the VC table is updated by creating a new pair of (associated) entries in both directions that corresponds to this VC and updating the corresponding fields. The CP then allocates local labels and updates the *Forward and Backward label* fields before forwarding the N\_SETUP message. (Note that the labels may be physical labels or logical labels for control only in the case of unidirectional connections) Otherwise, if the Forward Label field in the previous N\_SETUP contains a non-null field,<sup>9</sup> then the CP will locate the entry corresponding to this label in its VC table and use the Backward Label to update the output label field for the output port from which the N\_SETUP message came. If the residual bandwidth is smaller than the MIN value, then the required connection cannot be established and the connection setup is aborted; the aborting CP creates an N\_RELEASE message that is forwarded from it towards the source CP.<sup>10</sup>

Using VC traversal, when the N\_RELEASE message arrives at the first junction node, its CP will send an N\_DROP\_REPORT message of the VC traversal type towards the source CP, which upon receipt of this message will send an N\_RELEASE message to the remaining parts of the tree. Note that the source CP can determine the nature of the drop since the connection has not yet been completed, i.e., it knows that the drop is due to unsuccessful setup and not because of end point request.

When a CP receives an N\_SETUP message with the EXCHANGE function, it essentially performs the same operation as with the RESERVE, except that the CP is notified that the resource (bandwidth, etc.) has already been reserved for this connection, or that resources need not be reserved (third party setup), and will simply allocate and exchange labels by forwarding the

---

9. This indicates that the entry for this connection has already been created in this CP.

10. An interesting alternative is to continue the multicast connection setup until the end of the path, storing in the setup message the identities of those CPs for which the setup was successful. This information will then be returned to the source CP at the end of the traversal. Based on this information, the source will then have the option of whether to start the multicast session or not. This solution is not supported by the currently available user interface, and therefore we do not elaborate on its details.

N\_SETUP message with these labels. The CP should update the VC table so that the multicast branching effect is achieved. This is done by adding to the corresponding entry in the table additional  $vc_o$  and port entries. This will happen, at junction nodes of the tree on the second visit and on (see the example at the end of this section).

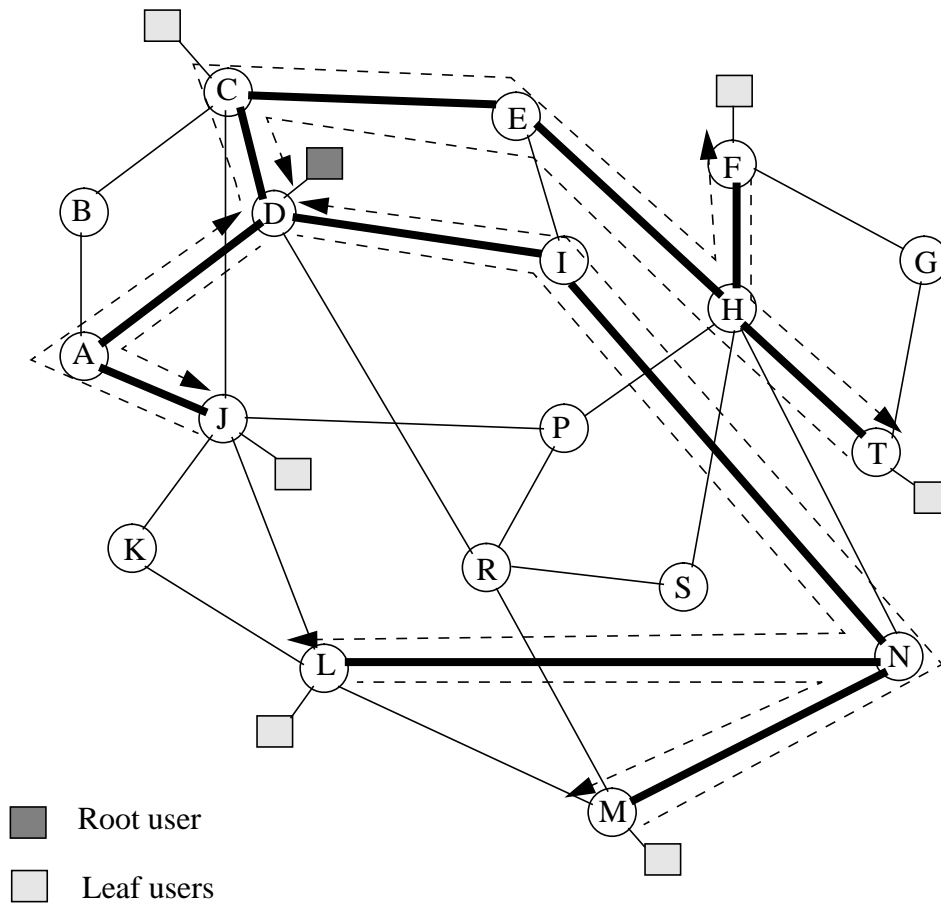
An END\_POINT function indicates that a user connected to this CP is a participant in the multicast. The CP must then check and verify the acceptance of the connection by the user before the message is forwarded. If the user does not accept the connection, the setup is aborted; the aborting CP creates an N\_RELEASE message that is forwarded from it towards the source CP as described earlier.

When the N\_SETUP message completes the traversal, it will be at the source CP. The MAX value contained in this message will be the final acceptable value for all nodes in the multicast tree. In order to inform all switches, it is necessary for the source CP to issue an N\_CHANGE message to notify all CPs of the final accepted bandwidth. The N\_CHANGE message is forwarded using the VC traversal mode. Each CP then adjusts the reservation to the final accepted value. The N\_CHANGE message is acknowledged by each CP containing one or more end-point users with the N\_CHANGE\_ACK message, which confirms and completes the multicast connection setup.

As an example, consider the network depicted in Figure 12 in which the root user attached to node *D* attempts to establish a multicast to leaf users attached to nodes  $\{C, F, J, M, L, T\}$ . This multicast connection is assumed to be unidirectional (as per UNI 4.0) although the setup procedure described here can be used to setup bidirectional connections as well. Node *D* computes a multicast tree (shown by bold links in the figure). To send the N\_SETUP message node *D* determines the traversal route (*D, A, J, A, D, C, E, H, F, H, T, H, E, C, D, I, N, L, N, M, N, I, D*) (shown by the dashed lines). As explained, the traversal route will start from the source (root) and end at the source also. Hence, all links are traversed exactly twice. Note also that some nodes are visited more than once (even the source node!) and will find the EXCHANGE function rather than RSRV\_\* (FWD, BWD, or BOTH) in visits other than the first. The list of functions for this message is given in Table 9.

#### **5.4.2 Step-by-step setup - multiple messages multicast setup**

Based on the best routes it determined, the host CP computes a traversal of the tree by several *connected paths*. Each of these paths ends at one of the destination CPs, and traverses part of the tree. Each is a simple path, i.e., it traverses a link exactly once. The paths are not necessarily disjoint. Notice that this approach easily supports current UNI signaling for multicast connection setup, as a separate setup message can be sent for each ADD PARTY message received from the root user. However, the multiple message setup is more general in the sense that it allows a setup message to add more than one end point as long as the path it traverses is a simple path. Note that it is possible to perform the step-by-step setup either synchronously or asynchronously. In the synchronous approach, a setup message must be acknowledged by the destination before the next one is sent. In the asynchronous approach, the setup messages can be transmitted in parallel without having to wait for the acknowledgment of other setup messages. In general, it is necessary for the setup messages to be identified with the set of



**Figure 12.** Example of a single message multicast setup

entries in the VC table corresponding to the multicast connection. This can be done via the Forward and Backward label fields in the N\_SETUP message as described below.

The N\_SETUP message used here is identical in structure to the one used in the single message case, with the same set of functions. All the N\_SETUP messages referring to the same connection will be identified based on the *Forward Label* contained in the previous N\_SETUP message. (In the case of the source CP, the call reference value supplied by the user will identify the appropriate connection.) In general, if the forward label is the *Null* label, then the CP does not have an entry for the connection and will need to create one; if the forward label is valid, then the CP will search its VC table and locate the appropriate input port and label, where the port is the one leading to its switch in the route and the label is the forward label contained in the N\_SETUP message. In order to handle the possibility of asynchronous parallel N\_SETUP messages, a special reserved label (ASYN) will be put in the forward label field. A CP receiving this label will use the backward label to identify the connection referred to by the setup message.

<b>Node</b>	<b>Link</b>	<b>Function</b>
D	DA	RF
A	AJ	RF
J	JA	EP;XC
A	AD	XC
D	DC	RF
C	CE	EP; RF
E	EH	RF
H	HF	RF
F	FH	EP; XC
H	HT	RF
T	TH	EP;XC
H	HE	XC
E	EC	XC
C	CD	XC
D	DI	RF
I	IN	RF
N	NL	RF
L	LN	EP; XC
N	NM	RF
M	MN	EP; XC
N	NI	XC
I	ID	XC

*NOTE:* RF = RSRV\_FWD, RB = RSRV\_BWD, R2 = RSRV\_BOTH,  
 XC = EXCHANGE, NU = NULL, EP = END\_POINT, ND = NODE\_ID.

**Table 9.** Example of a single N\_SETUP message for multicast

Generally, the setup operation is similar to the unicast setup with the modification that a link might be used just for advancing the message without setup and reservation operations even the first time this message traverses the link. Note that updating the VC table is similar to the case of a single message multicast since an N\_SETUP message referring to the same connection may arrive at the CP more than once. When a message arrives at the CP with an RSRV\_\* function (probably the first time), the VC table update is simple (creating a new entry in the table and updating its fields). Every other arrival of a message for the same connection to the CP will trigger the CP to update the corresponding entry of the VC table, by adding extra  $vc_o$

and port fields for the corresponding  $vc_i$  field. Note that associated entries (logical labels for unidirectional connections) are also created or updated.

For each path, the N\_SETUP\_ACK message progresses backwards as in the unicast case. Upon receiving N\_SETUP\_ACK messages from all paths, the source CP lets the source host start its communication with the destination hosts using the minimum value among all the MAX values it received from the different paths. Note that the MAX values received from different paths can be different. To reclaim unused bandwidth (the difference between the MAX values received on a path and the MIN value that is used), the source CP starts a procedure for decreasing the reserved unused bandwidth. This is done by the source CP sending an N\_CHANGE message that traverses the entire multicast tree (in the VC traversal mode) and causes its CPs to update their tables accordingly. Note that the N\_CHANGE message has another functionality that is related to connection maintenance, as explained in Section 5.2.

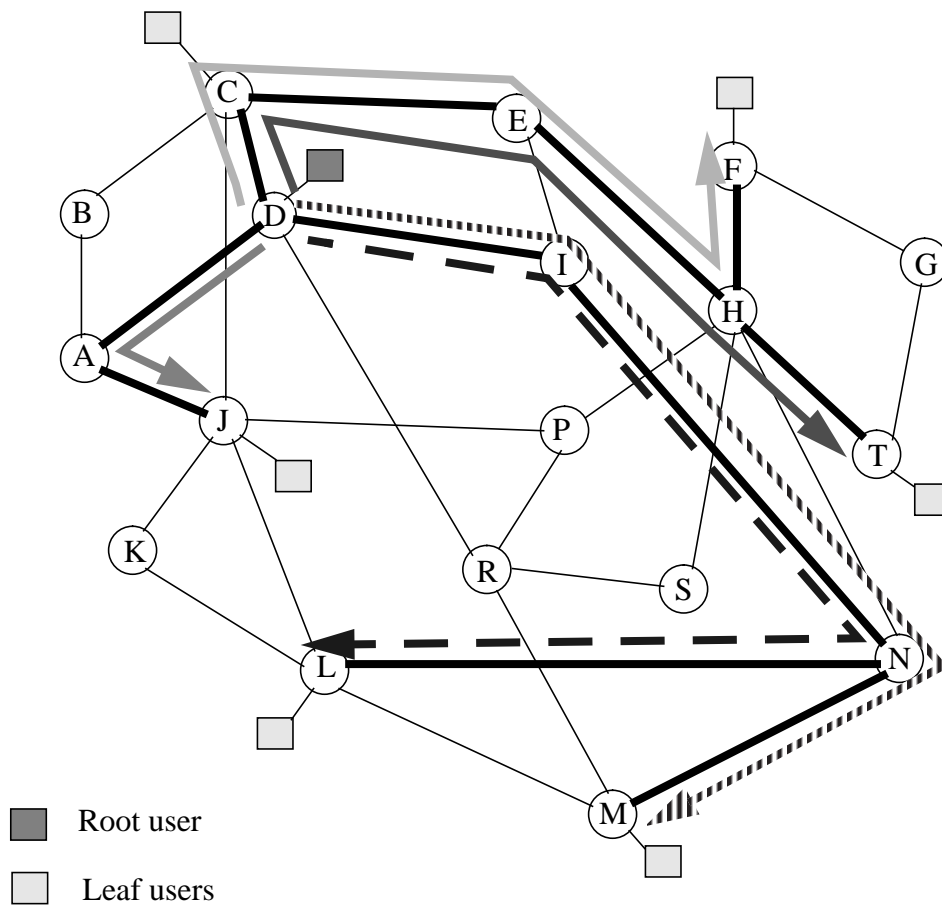
Figure 13 depicts the connection setup process for the same network and multicast tree as in Figure 12. The simple paths chosen by the source node D are (D,A,J), (D,I,N,M), (D,I,N,L), (D,C,E,H,F), and (D,C,E,H,T), and are shown in the figure with different shaded arrows. For simplicity, we assume the synchronous approach for the multiple message setup in this example. As explained, some links (e.g., (D,A)) are traversed by a single N\_SETUP message while some are traversed by more than one N\_SETUP messages (e.g., (D,I)). The list of functions for each of the N\_SETUP messages is depicted in Table 10. The N\_SETUP\_ACK message, for each of the paths will traverse the same route back from last node on the path to the source D.,

### 5.4.3 Discussion

The advantages and disadvantages of each of the above multicast setup methods are obvious. With a single message setup, the messages are long, since a single message contains the traversal of the whole tree. With multiple messages setup, the messages are shorter, since each message contains the traversal of part of the whole tree. Since only simple paths are used with multiple messages setup (which implies an increased number of setup messages), then each of the individual setups is simpler than the setup with a single message. In either case, N\_CHANGE messages are necessary to free unused reserved bandwidth. Also, asynchronous step-by-step setup can proceed faster than the single message setup.

### 5.4.4 OPENET and UNI signaling for multicast connection setup

We first define some terms to identify the differences in functions performed at each of the switch CPs involved in a multicast connection setup. Such differences do not exist in a unicast connection setup. A CP is said to represent a *Point-to-Multipoint (PMP) Node* if cells destined to at least two destinations (leaves) in the multicast tree need to traverse the switch controlled by the CP. In addition, a PMP-Node is said to be a *Last Common (LC)-Node* if for a given (partially set up) tree and for a given new destination, the next switch along the tree to the destination is not a PMP-Node. Finally, a CP is said to represent a *Single Party (SP)-Node* if, for a given multicast tree, the cells for only one destination need to traverse the switch controlled by



**Figure 13.** Example of a multiple message multicast setup

the CP. As an example, for the multicast tree described in Figures 12 and 13, switches C,E,I,H,N are PMP-nodes and switches A,J,L,M,F,T are SP-nodes. Also, switch N is an LC-node with respect to the destination user attached to node M, and so forth. In order to describe the OPENET mechanism for link failure notification, it is also useful to define a *Branching (BR)-Node*. A CP is said to represent a BR-Node with respect to a given multicast connection if the degree of the node in the tree representation of the multicast connection has a degree greater than 2. Switches D,C,H,N are BR-nodes in the same example above. Note that a LC-Node is also a BR-Node, but not necessarily the opposite.

Using UNI 4.0 signaling, the Step-by-Step setup method can be used directly to setup multicast connections. The setup of a multicast connection is accomplished by making connections to each of the endpoints one by one. The UNI messages used for this type of setup are the SETUP, ADD PARTY, ADD PARTY ACKNOWLEDGE, and ADD PARTY REJECT. In addition, the DROP PARTY and DROP PARTY ACKNOWLEDGE messages are used to partially clear a multicast connection.

<b>Node</b>	<b>Link</b>	<b>Function</b>
D	DA	RF
A	AJ	RF
J	J-LEAF	EP
D	DI	RF
I	IN	RF
N	NM	RF
M	M-LEAF	EP
D	DI	XC
I	IN	XC
N	NL	RF
L	L-LEAF	EP
D	DC	RF
C	CE	EP; RF
E	EH	RF
H	HF	RF
F	F-LEAF	EP
D	DC	XC
C	CE	XC
E	EH	XC
H	HT	RF
T	T-LEAF	EP

**Table 10.** Example of multiple N\_SETUP messages

For a multicast connection, the root and leaf terminals, and the network side UNI at the root and leaf maintain two separate machines: a point-to-multipoint (PMP) state machine which maintains states for each participant in a call and a link-state machine which maintains the Q.93B states. The states in the PMP state machine are called party states: “NULL” (P0), “AP-INITIATED” (P1), “AP-RECEIVED” (P2), “DP-INITIATED” (P3), “DP-RECEIVED” (P4), and “ACTIVE” (P5). The states in the link-state machine are as defined in UNI 3.0. In the following subsections, we will describe how to use the OPENET messages defined earlier to accomplish the step-by-step setup.

#### 5.4.4.1 Setup of the first party

The setup of the first party is initiated by the user at the root B-TE using a UNI SETUP message containing a Broadband bearer capability information element indicating point-to-multi-point in the user plane connection configuration field, and an Endpoint reference information element with value 0. Since this is a first party setup, all the nodes along the path are by definition SP-Nodes. The signaling exchange among the CPs are the same as in the unicast case, with the only exception being the additional Endpoint reference IE carried in the SETUP and CONNECT messages.

#### 5.4.4.2 Adding a party

The calling party (Root) can initiate the addition of a party by sending an ADD PARTY message, provided that the link is in the Active (U10) link-state. After sending the ADD PARTY message, the user starts timer T399. The ADD PARTY message must have the same call reference value as specified in the initial setup of the call to which the party is to be added. The message also contains an endpoint reference (*ER*) value. The party state for the new endpoint ( $ER = n$ ) will change to AP-Initiated (P1) after the transmission of the ADD PARTY message. The connection identifier (VPCI/VCI) used for the new party is the same as for the original call, and is not indicated in the ADD PARTY message. The QoS, Bearer capability, and ATM user cell rate for the new party are the same as for the original call and are not indicated in the ADD PARTY message. Therefore, it is necessary for the S\_CP to maintain the appropriate information pertaining to the call reference value.

In general, a route to the new endpoint leaf B-TE will contain zero or more PMP-Nodes, one LC-Node, and zero or more SP-Nodes. For illustration purposes, we assume that adding the new party involves one PMP-Node, one LC-Node, and one SP-Node. Notice that the function performed by the N\_SETUP message at the CPs depends on whether the node represents a PMP-Node, an LC-Node, or an SP-Node. The type of node that the CP represents (with respect to a given connection/destination) can be derived from the information contained in the VC table. For example, since ADD PARTY messages are sent only by the root, the S\_CP receiving an ADD PARTY message must represent either a PMP-node or an LC-node. If the entry in the VC table at the S\_CP associated with the given connection already contains the output port and label for the route selected, then it is a PMP-node; otherwise, it is an LC-node. At an I\_CP or the D\_CP, if the received N\_SETUP message contains both forward and backward labels that already exist in its VC table, then it is either a PMP-node or an LC-node; otherwise, it is an SP-node. If not an SP-node and the entry in the VC table for the connection also contains the label for the output port specified in the N\_SETUP message for this switch, then it is a PMP-node; otherwise, it is an LC-node.

If the S\_CP determines that access to the requested service is authorized and available, it will change the party state (for  $ER = n$ ) to AP-Received (P2) and transmit an N\_SETUP message to the next node along the path. Since the S\_CP represents a PMP-Node, both the VC labels and the bandwidth have already been allocated for this connection at this switch. At the I\_CP, since it represents an LC-Node, the N\_SETUP message will trigger the *output* label<sup>11</sup> and bandwidth

---

11. Actually, the output label will be allocated when the N\_SETUP\_ACK message is received by the I\_CP.

allocation process as in the unicast case. (The input label has already been allocated at this node.) At the D\_CP, since it represents an SP-Node, the N\_SETUP message will result in the allocation of both input and output labels as well as the bandwidth for this connection. Also, the D\_CP will send a SETUP message with end point reference  $n$  to the leaf user B-TE, and starts timer T303. If the user accepts the call, it will respond with a CONNECT message with  $ER = n$ , start timer T313, and wait for a CONNECT ACKNOWLEDGE message. The D\_CP will complete the adding of the party by sending an N\_CONNECT message back towards the root using the VC traversal forwarding mode. Note that the signaling messages in the OPENET architecture uniformly support unicast and step-by-step multicast setup, i.e., the same sequence of control messages is used for unicast connection setup, first party setup, and add party setup of multicast connections.

## **5.5 Multicast connection clearing**

Take down operations are needed to release the resources upon the termination of a multicast connection or the dropping of a party from the connection. In addition, take down operations are needed to release the resources upon failures which cause the disconnection of a part of the multicast connection.

### **5.5.1 End of connections**

Releasing a multicast connection by the source is done using the N\_RELEASE message as in a unicast connection. That is, the N\_RELEASE message traverses the multicast tree using the VC traversal mode and releases the bandwidth and labels of the connection. In the other direction, a different process is used to drop a party from the connection. According to this process, the party sends an N\_RELEASE message on the backward direction of the multicast connection using the VC traversal mode. This message will release the bandwidth and the labels up to the corresponding LC node. The LC node, upon receipt of this message, will send an N\_DROP\_REPORT message of the VC traversal type to the source. This message contains the end point reference of the party node that started the drop operation. When this message arrives at the source node, it notifies the UNI that the corresponding party was dropped, using the end point reference of that party. The source node can issue a drop party operation by sending an N\_DROP\_REQUEST message which contains a list of end point reference IDs of the corresponding parties. This message is of the VC traversal type, and it traverses the multicast connection and arrives at all destination nodes. All destination nodes whose end point reference is included in this message will start a drop party process as described before. As a result, an N\_DROP\_REPORT message will be delivered to the source for each dropped party, completing the drop operation.

### **5.5.2 Failures**

As in the unicast connection, when a CP detects a failure, it determines which connections are affected by this failure. In case the connection lost its path to the source (root), the CP sends an N\_RELEASE message over all other local links it traverses. Such N\_RELEASE messages will be sent by all the CPs that are adjacent to the failure. As a result, the disconnected part(s) of the multicast connection will be released as before. Similarly, if the failure has resulted in a situation where the node has no longer any downstream links emanating from it (including all

end-point leaves such as maybe the CP itself), it will send an N\_RELEASE message in the direction of the source. This N\_RELEASE may carry an end-point reference ID in case the falling link was a UNI end-point connection. Otherwise this field is set to null. This N\_RELEASE message will traverse the connection in the source direction and release the bandwidth and labels for this connection. The first BR-node that receives this message will convert this message into an N\_DROP\_REPORT message and send it towards the source. This message contains the same end-point reference ID of the received N\_RELEASE and the node ID of the BR-node along with the link ID from which the N\_RELEASE message was received. When the source receives this message it can determine (using the multicast connection tree structure) which nodes have been dropped as a result of the failure. Note that if the BR-node is the one adjacent to the failure (and does not become a leaf after that failure as discussed above), it will send N\_DROP\_REPORT to its upstream link in the direction of the source.

### **5.5.3 OPENET signaling for multicast connection clearing**

Using UNI 4.0 signaling, the root or leaf user of a multicast connection can initiate partial connection clearing by sending DROP PARTY messages. In the case that there is only one party at the interface, a RELEASE message will be used. We describe in this section the OPENET mechanisms that can be used to support these UNI messages. At the root, the user will initiate dropping a party by sending a RELEASE or DROP PARTY message. The RELEASE message is sent if all other parties belonging to the same call on the interface are in the P0, P3, or P4 party-states. A DROP PARTY message is used to initiate party clearing when there are other parties to the call on this interface in the P1, P2, or P5 party-states.

#### **5.5.3.1 Drop party by root**

In this case, the root user attempts to drop a party while other parties are still active. It transmits a DROP PARTY message across the UNI with end point reference IE indicating which end point should be cleared. The source CP will then forward an N\_DROP\_REQUEST message onto the multicast tree using the VC traversal forwarding mode. Note that this message will actually arrive at all leaf nodes, but only that with a matching End Point Reference will respond. The destination CP also sends a RELEASE across the UNI following the same UNI clearing procedure for unicast connections. (We assume that there is only one party at each leaf UNI.) The D\_CP with the matching End Point Reference will initiate a clearing procedure by sending an N\_RELEASE message upstream, back towards the root with the end point reference value for the dropped party included in the message. This N\_RELEASE message will release the bandwidth and labels used on all SP-nodes along the path up to the LC node which will forward an N\_DROP\_REPORT message. The N\_DROP\_REPORT message will not release bandwidth and labels since there are other parties still in this connection. When the N\_DROP\_REPORT message arrives at the S\_CP, it will send a DROP PARTY ACKNOWLEDGE message across the interface and complete the drop party.

#### **5.5.3.2 Release by root**

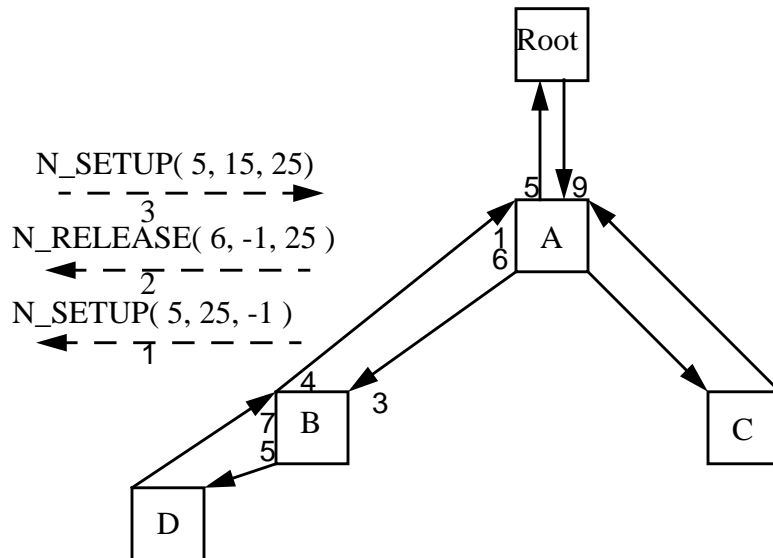
A RELEASE message is sent by the root to clear the last party of a multicast connection. The procedure used is similar to the clearing procedure for unicast connections, with the exception that the party states need to be reset to null also.

**5.5.3.3 Drop party by leaf**

When a leaf user wishes to be cleared from a multicast connection, it will send a RELEASE message across the UNI to the network.<sup>12</sup> The D\_CP will respond with a RELEASE COMPLETE message and return to the null link state and null party states as in the unicast case. The D\_CP also forwards an N\_RELEASE message towards the root with an End Point Reference value indicating the party requesting the clearing. At the LC-node, the CP will forward an N\_DROP\_REPORT message towards the root indicating the appropriate End Point Reference. At the S\_CP, a DROP PARTY message is sent across UNI with the End Point Reference carried in the N\_DROP\_REPORT message. The party clearing is complete when the root user responds with a DROP PARTY ACKNOWLEDGE message.

**5.5.4 Consistency check**

As in the unicast connection consistency check, each CP receiving a VC traversal message must check for consistency of the label indicated in the message. Another consistency check is needed during the setup phase as described in the example shown in Figure 14. Here we assume a single message multicast setup. When node A receives an N\_SETUP message back from B after an N\_RELEASE message for that connection has been sent by A to node B, it discovers that the connection was released, since the forward label included in the message (label 25) doesn't exist anymore. Note that backward consistency checks cannot be used in this case.



**Figure 14.** Setup consistency in the forward direction.

The N\_DROP\_REPORT and the N\_DROP\_REQUEST packets are of the VC traversal type.

12. This is assuming that only one party exists at the interface; therefore, RELEASE is used rather than DROP PARTY.

## **5.6 Multicast Connection Maintenance**

Similar to unicast connections, the source CP of a multicast connection periodically sends N\_REFRESH messages for each established multicast connection. This message is forwarded according to the VC traversal mode. Changing the reserved bandwidth of a multicast connection is also similar to unicast connections. Only the source of the multicast connection can send an N\_CHANGE message to increase or decrease the bandwidth of the connection. This message includes the minimum and maximum bandwidth desired for this connection and a refresh period. The N\_CHANGE message is acknowledged by N\_CHANGE\_ACK messages that contains the final negotiated values from each of the parties. If all parties agree on the final values, the change is completed. Otherwise, another N\_CHANGE message with the least final value among all parties has to be sent. However, this time the least final value is guaranteed on each link of the multicast connection as every link has reserved at least that value.

## **5.7 Distribution Tree Setup**

As described in Section 3.3, the distribution tree may be constructed using either a pivot-based or a load-balanced tree-based approach. In the pivot-based case, the distribution tree consists of a root CP (the pivot), a destination tree, and a source tree. In the load-balanced tree based case, each node behaves as if it were the pivot, such that cells arriving at any node will be distributed over all outgoing links, except the one from which they come.

In general, the sets of source CPs and destination CPs may be different. However, in the distribution tree that is used to distribute the utilization updates, each of the two sets contain the set of all CPs in the system. In this section, we concentrate on the setup of the distribution tree that is used for utilization update distribution.

### **5.7.1 Load-balanced tree based setup**

In the load-balanced tree based distribution tree setup, an arbitrary CP will be chosen as the root of the tree. A tree with this CP as root is computed using the route computation algorithm described in Section 6.4.2. In a load-balanced tree based distribution tree, every switch will multicast any incoming cell to all output ports except the one in which the cell came from. In the following, we describe the single message setup of a load-balanced tree based distribution tree.

To set up the distribution tree, the root CP first creates an N\_SETUP message with a complete traversal of the computed tree. The N\_SETUP message will be forwarded from one CP to another on the tree, according to the source route specified in the initial message. At each CP along the traversal path, the VC labels and bandwidth are allocated and reserved, if not already done. Since each link is traversed exactly twice, the VC labels are allocated for both directions after the link has been traversed the second time. Each CP also updates the Forward and Backward Traversal Label fields in the N\_SETUP message before forwarding the message to the next CP. In each of the nodes, the VC tables are updated using the same algorithm since there is no pivot node involved. This update is indicated in the N\_SETUP by the load-balanced tree

function (BALANCE) defined in Section 3.3.1. Note that while the BALANCE function dictates programming the switch in such a way that the distribution tree operation is conducted (a cell is sent over all other links), for control purposes, the CP keeps its local tables in the same way all trees (source, destination, pivot-based) are maintained, with one direction leading to the root and the other toward the leaves. This is done in order to allow the root maintaining the tree using the normal VC traversal type tree-maintenance messages such as N\_CHANGE, N\_CHANGE\_ACK, and N\_REFRESH as well as using normal connection clearing procedures.

As an example of the load-balanced tree based setup, consider the setup of the same utilization tree shown in Figure 2. Assume that node E is chosen as the root, and the computed traversal route is  $\{E, A, D, A, E, C, F, C, E, B, E\}$ . The status of the switch VC tables and the forward and backward label fields in the N\_SETUP message as it traverses the route are shown in sequence in Table 11. All the associated entries for the given distribution tree are given in the VC table entry column. By convention, we use port 0 to indicate the link to the node's CP. The other ports are named using the pair of nodes at either ends of the links. For example, in node A, the port to node D is named AD, while in node D the port to node A is named DA, and so on. The labels allocated by a node are denoted by small letters, e.g., at node A the labels are  $a_0, a_1, a_2, \dots$ , etc.

Node	VC Table Entries	FWD Label	BWD Label
E	$(0, e_0) \rightarrow (EA, \text{null})$ $(EA, e_1) \rightarrow (0, e_0)$	null	$e_1$
A	$(0, a_0) \rightarrow (AE, e_1)(AD, \text{null})$ $(AE, a_1) \rightarrow (0, a_0)(AD, \text{null})$ $(AD, a_2) \rightarrow (0, a_0)(AE, e_1)$	null	$a_2$
D	$(0, d_0) \rightarrow (DA, a_2)$ $(DA, d_1) \rightarrow (0, d_0)$	$a_2$	$d_1$
A	$(0, a_0) \rightarrow (AE, e_1)(AD, d_1)$ $(AE, a_1) \rightarrow (0, a_0)(AD, d_1)$ $(AD, a_2) \rightarrow (0, a_0)(AE, e_1)$	$e_1$	$a_1$
E	$(0, e_0) \rightarrow (EA, a_1)(EC, \text{null})$ $(EA, e_1) \rightarrow (0, e_0)(EC, \text{null})$ $(EC, e_2) \rightarrow (0, e_0)(EA, a_1)$	null	$e_2$

**Table 11.** Load-balanced tree based utilization tree setup example

Node	VC Table Entries	FWD Label	BWD Label
C	$(0, c_o) \rightarrow (CE, e_2)(CF, \text{null})$ $(CE, c_1) \rightarrow (0, c_0)(CF, \text{null})$ $(CF, c_2) \rightarrow (0, c_0)CE, e_2)$	null	$c_2$
F	$(0, f_o) \rightarrow (FC, c_2)$ $(FC, f_1) \rightarrow (0, f_o)$	$c_2$	$f_1$
C	$(0, c_o) \rightarrow (CE, e_2)(CF, f_1)$ $(CE, c_1) \rightarrow (0, c_0)(CF, f_1)$ $(CF, c_2) \rightarrow (0, c_0)CE, e_2)$	$e_2$	$c_1$
E	$(0, e_o) \rightarrow (EA, a_1)(EC, c_1)(EB, \text{null})$ $(EA, e_1) \rightarrow (0, e_0)(EC, c_1)(EB, \text{null})$ $(EC, e_2) \rightarrow (0, e_0)(EA, a_1)(EB, \text{null})$ $(EB, e_3) \rightarrow (0, e_0)(EA, a_1)(EC, c_1)$	null	$e_3$
B	$(0, b_0) \rightarrow (BE, e_3)$ $(BE, b_1) \rightarrow (0, b_0)$	$e_3$	$b_1$
E	$(0, e_o) \rightarrow (EA, a_1)(EC, c_1)(EB, b_1)$ $(EA, e_1) \rightarrow (0, e_0)(EC, c_1)(EB, b_1)$ $(EC, e_2) \rightarrow (0, e_0)(EA, a_1)(EB, b_1)$ $(EB, e_3) \rightarrow (0, e_0)(EA, a_1)(EC, c_1)$	-	-

**Table 11.** Load-balanced tree based utilization tree setup example

The distribution tree is completely set up when the N\_SETUP message returns to the root node E for the last time. Although node E is designated as the root node, it uses the same algorithm to create VC table entries for the tree as every other node. The only additional task at the root node is to compute the traversal route and create the initial N\_SETUP message.

### 5.7.2 pivot based Setup

In the pivot based distribution tree setup, a CP is first chosen as the pivot. The pivot CP then computes source and destination trees using the route computation algorithm described in Section 6.4.2. In general, these two trees can be set up independently and then merged at the pivot to form a distribution tree. However, if the same links are used in both trees, their setup can be done simultaneously. This is because the setup messages traverse the same links. Note that the setup of the destination tree is essentially a multicast connection setup, where the multicast contains all CPs in the system. Therefore, we can use either the single message or the multiple messages multicast setup approach as described in Section 5.4. In the following, we

describe the single message setup of a pivot based distribution tree in which the set of sources and destinations are the same, and where the source and destination trees use the same links.

The setup of the distribution tree in this case is essentially a multicast setup (with physical labels in both directions) with additional operations at the pivot of the tree. To set up the distribution tree, the root CP first creates an N\_SETUP message with a complete traversal of the common source and destination tree. The N\_SETUP message will be forwarded from one CP to another on the tree according to the source route specified in that message. At each CP along the traversal path, the VC labels and bandwidth are allocated and reserved, if not already done. Since each link is traversed exactly twice, the VC labels are allocated for both directions after the link has been traversed the second time. Each CP also updates the Forward and Backward Traversal Label fields in the N\_SETUP message before forwarding the message to the next CP. Additional operation is performed at the pivot of the tree using the pivot type function.

As an example of the pivot based setup, consider the setup of a pivot based utilization tree shown in Figure 2. Assume that node E is the pivot and the computed traversal route is  $\{E, A, D, A, E, C, F, C, E, B, E\}$ . The status of the VC tables and the forward and backward label fields in the N\_SETUP message as it traverses the route are shown in sequence in Table 12.

Note that the source and destination trees are completely set up when the N\_SETUP message returns to the pivot node E at the last time and they are already joined together at the pivot node E. The manner in which the VC table entries are created at the pivot node is different from those at the other nodes, since input to each distribution tree port of the pivot node E will be multicast to all distribution tree ports, including port 0 to the local CP. In the case of a utilization tree, the bandwidth reservation can be based on a fixed value instead of a range from MIN to MAX. In this situation, once the N\_SETUP message finishes the traversal, the distribution tree setup is complete.

Node	VC Table Entries	FWD Label	BWD Label
E	$(0, e_o) \rightarrow (EA, \text{null})$ $(EA, e_1) \rightarrow (0, e_o)(EA, \text{null})$	null	$e_1$
A	$(0, a_o) \rightarrow (AE, e_1)$ $(AE, a_1) \rightarrow (0, a_o)(AD, \text{null})$ $(AD, a_2) \rightarrow (AE, e_1)$	null	$a_2$
D	$(0, d_o) \rightarrow (DA, a_2)$ $(DA, d_1) \rightarrow (0, d_o)$	$a_2$	$d_1$

**Table 12.** Pivot-Based Utilization Tree Setup Example

Node	VC Table Entries	FWD Label	BWD Label
A	$(0, a_o) \rightarrow (AE, e_1)$ $(AE, a_1) \rightarrow (0, a_0)(AD, d_1)$ $(AD, a_2) \rightarrow (AE, e_1)$	$e_1$	$a_1$
E	$(0, e_o) \rightarrow (EA, a_1)(EC, \text{null})$ $(EA, e_1) \rightarrow (0, e_0)(EA, a_1)(EC, \text{null})$ $(EC, e_2) \rightarrow (0, e_0)(EA, a_1)(EC, \text{null})$	null	$e_2$
C	$(0, c_o) \rightarrow (CE, e_2)$ $(CE, c_1) \rightarrow (0, c_0)(CF, \text{null})$ $(CF, c_2) \rightarrow (CE, e_2)$	null	$c_2$
F	$(0, f_o) \rightarrow (FC, c_2)$ $(FC, f_1) \rightarrow (0, f_o)$	$c_2$	$f_1$
C	$(0, c_o) \rightarrow (CE, e_2)$ $(CE, c_1) \rightarrow (0, c_0)(CF, f_1)$ $(CF, c_2) \rightarrow (CE, e_2)$	$e_2$	$c_1$
E	$(0, e_o) \rightarrow (EA, a_1)(EC, c_1)(EB, \text{null})$ $(EA, e_1) \rightarrow (0, e_0)(EA, a_1)(EC, c_1)(EB, \text{null})$ $(EC, e_2) \rightarrow (0, e_0)(EA, a_1)(EC, c_1)(EB, \text{null})$ $(EB, e_3) \rightarrow (0, e_0)(EA, a_1)(EC, c_1)(EB, \text{null})$	null	$e_3$
B	$(0, b_o) \rightarrow (BE, e_3)$ $(BE, b_1) \rightarrow (0, b_o)$	$e_3$	$b_1$
E	$(0, e_o) \rightarrow (EA, a_1)(EC, c_1)(EB, b_1)$ $(EA, e_1) \rightarrow (0, e_0)(EA, a_1)(EC, c_1)(EB, b_1)$ $(EC, e_2) \rightarrow (0, e_0)(EA, a_1)(EC, c_1)(EB, b_1)$ $(EB, e_3) \rightarrow (0, e_0)(EA, a_1)(EC, c_1)(EB, b_1)$	-	-

**Table 12.** Pivot-Based Utilization Tree Setup Example

Once the N\_SETUP message returns to the root CP for the final time, there is a need for an additional phase to let all CPs know that the setup of the distribution tree has been completed,

and they can start to use it to distribute utilization updates. This is done by the pivot CP sending an N\_CHANGE message to all CPs on the destination tree using the forward VC traversal mode. In the case that a range of bandwidth reservation is specified in the N\_SETUP message, the N\_CHANGE message also serves to release any unused bandwidth in case the MAX value contained in the final N\_SETUP message is lower than the bandwidth already reserved on some links of the tree.

## **5.8 Distribution tree reset**

Once a distribution tree has been set up, it will be used to disseminate utilization data to all CPs in the network. In case of link failures or node failures that cause the distribution tree to be disconnected, it is necessary to either “repair” it or to take it down and start a new one. Because of routing considerations, it is easier to take down the disconnected tree and start a new one. This process is described in Section 6.2.

## **5.9 Third party connection setup**

Although UNI 3.0 does not support third party connection setup, OPENET can provide this capability by the use of the function field in the N\_SETUP message. Assume a manager host connected to a manager CP (M\_CP), a source host connected to a source CP (S\_CP), and a destination host connected to a destination CP (D\_CP). The manager host can set up a connection between the source host and the destination host using a third party setup request to its M\_CP. The M\_CP computes a route from the S\_CP to the D\_CP using the route computation algorithm and constructs a setup message which contains a route from the M\_CP to the S\_CP, and the computed route from the S\_CP to the D\_CP. It uses the EXCHANGE function along the route from the M\_CP to the S\_CP to exchange logical labels only. These labels are needed by the M\_CP to maintain the connection, as we assume that the M\_CP continues to maintain this connection. Note that, if labels in both directions are logical, the node will perform checking and pass on functions only, so that messages such as N\_CHANGE will affect the route from the S\_CP to the D\_CP only.<sup>13</sup>

## **6.0 Topology/Utilization Update and Call Routing**

---

The topology and utilization update mechanism is designed to maintain a replicated database at all CPs so as to enable each CP to compute an effective route for new arriving calls or

---

13. Note that the manager CP (and all other CPs) is normally computing routes from itself to all other nodes using the same bandwidth requirement for the whole path. Therefore, a more efficient alternative for the third party setup might be the following procedure. The M\_CP only calculates a route to the S\_CP (with no bandwidth requirements) and includes it as part of the N\_SETUP with the EXCHANGE function. Another function COMPUTE (defined here for the first time) instructs the S\_CP to compute the remainder of the path (from the source switch to the destination with the bandwidth requirements as included in the received N\_SETUP), appending it to the source route field of the N\_SETUP, and continuing the N\_SETUP forwarding accordingly. In this design option the M\_CP triggers and maintains the connection, while the S\_CP performs the critical part of the route calculation. Hence, route computation is used in the usual way.

whenever rerouting is necessary. The database must therefore include all the relevant information for such routing and potentially for other network control functions. Examples of information items included in the data base are: the set of switches; the set of communication links and the manner in which they are interconnected; the ID of CPs, link and switch capacities; link and switch current utilization levels, etc. Other information, which we will term *attributes*, can also be included. These might be information regarding cost, length or security properties of links, inherent properties of switches (for example, multicast throughput limitations for certain switches), state information for CPs, reachable addresses or domains (if no other directory services support this information), etc. In this section, we will focus only on the topology database information which describes the network's switches and the communication links that connect them.

The contents of the database are derived from local information and from information shared with other CPs. Local information is that which is associated with the local CP cluster and is gathered directly from the CPs and switch's local tables, as well as from its cluster switches and their attached links. The rest of the database contains similar information regarding remote CPs, switches and links, gathered from other CPs in a manner described below. Note, however, that the local information is generally more detailed than the remote one since it may contain information not shared with remote CPs.

The update process is employed in order to synchronize all the replicas of the database at all CPs with the same concurrent information. Each CP is responsible to originate update messages whenever new topology information is detected in its local cluster or old information regarding local items becomes obsolete.

In OPENET, the contents of the database are logically divided into quasi-static and dynamic portions, according to the expected rate of its change. Quasi-static information is that which changes quite infrequently, such as when a failure occurs or a planned network reconfiguration takes place. Examples of quasi-static information items are link and nodal operational states (i.e, up or down), link error rates, link and switch capacities, link security level, link and switch service policy, and links and switch addresses. Dynamic items are those which change frequently, for example, with the establishment, behavior change, and termination of connections. Dynamic data, therefore, have a frequent and continuous impact on the ability of links and switches to accommodate additional calls or rate increases for existing calls. Examples of dynamic information items are changes in switch and link loads of certain traffic types.

Topological changes, while being quasi-static and quite rare, have a major effect on the connectivity of CPs and therefore make the task of building a reliable database update mechanism quite complex. It is clearly much simpler to handle information updates in a fixed topology network and to improve the speed and efficiency of the update mechanism in such an environment.

Our "optimistic approach" to network control optimizes the system for the most frequent events, and trades performance for simplicity in the case of exceptional and rare events. Therefore, for performance reasons, we have separated the mechanisms for exchanging quasi-static information (*topology update*) from the mechanism of exchanging dynamic information (*utilization update*). For the utilization update, which imposes fast and frequent update require-

ments, we suggest a mechanism which is optimized for both update-latency and computational overhead and assumes a fixed topology. Using this assumption, we devise a lightweight procedure that enables the CP to sustain a higher rate of updates than would otherwise be possible at an extremely low latency. Compared to the strict performance requirements of the utilization update, we relax the requirements for updating the quasi-static information. We adopt a conservative and standard mechanism that handles these updates and can also cope with topological changes. Since at times of such changes the utilization update mechanism may cease to work, the topology update mechanism carries utilization data along with the topology update as a backup mechanism.

In the normal mode of operation, the quasi-static information is synchronized, and the utilization update mechanism is free to assume a fixed topology. With such a setting we use native ATM multicasting to accomplish the task of utilization update (further details are given below). This update functions properly as long as no change occurs that disrupts the multicast set-up. Thus, minor topological changes (those which do not impact the utilization update multicast connections) will have no effect on the utilization update mechanism. The topological update mechanism will ensure that these changes are disseminated in a timely manner to all CPs.

When a major topological change occurs that disrupts the utilization update mechanism (e.g., switch failure), more massive operations must take place. First, the topology update mechanism is activated and disseminates the updated information to all CPs. Once this is done, the utilization update mechanism must be set-up anew, and only then can normal operation resume. Note that for this to work, termination detection of the topology update is necessary. To minimize the time in which current utilization information is unavailable at the nodes, the topology update messages include utilization information in addition to the topology information it usually carries.

Since OPENET is designed as a PNNI extension, we reuse the existing PNNI components as much as possible. Therefore, we use the standard PNNI topology update methods which in turn leverages the topology update design of the OSPF (Open Shortest Path First) used to assist IP routing in router-based networks [9] [14] [7]. It is implemented using the exchange of packets between neighboring nodes utilizing a reliable hop-by-hop flooding algorithm for the propagation of the topology database items. It is not optimized in terms of latency and overhead.

The use of a flooding algorithm implies that update messages are received in duplicates from all neighbors entailing redundant overhead. The specific flooding algorithm used by PNNI (and OSPF) also entails a store-and-forward operation at every hop. This forces the protocol to read with no delays each received update in order to check whether forwarding is required. This excludes the possibility of processing a large number of duplicates in batch (doing so will increase the latency of the update). Note, however, that since ATM switches allow point-to-point and multicast connections, it is possible, at least theoretically, to improve this latency and excess overhead. On the other hand, PNNI provides the topology update function through a tested and standardized method. It uses essentially the same basic scheme which has been successfully employed by the Internet for over 20 years. This advantage overcomes the fact that

there might be theoretical improvements to the OSPF update algorithm [6]. Moreover, even the best known topology update algorithms (which by definition must operate under topological changes) have a considerable higher latency and overhead than our proposed utilization update algorithm, which makes the argument of separating both still valid.

Finally, our architecture uses a network-wide spanning distribution tree for the task of utilization update. The same distribution tree can be used by all nodes to broadcast local changes to all others using unicell updates. Such a scheme delivers only a single copy of the information to each CP, utilizes the switch hardware multicast where possible and uses a small and fixed amount of VCIs. Since no forward operation is required at the CP, it also allows batch processing of multiple utilization messages. On the other hand, such a distribution tree needs to be initially set-up and subsequently maintained (repaired) in reaction to topological changes which impact its connectivity or network-wide span. A full description of the utilization update can be found below.

### **6.1 Highlights of PNNI topology update**

In the following, we sketch the PNNI topology update mechanism (see [3] for details). Each CP triggers periodically or upon a local change topology updates which are marked by increasing sequence numbers (4 bytes), each by one increment from the one previously used. Separate sequence numbers are associated with each topological item (PTSE). These updates also carry an age field (4 bytes) which relates to the time elapsed since the source originated this information. Each PTSE consists of one or more topology information groups (TIG). These TIGs carry information about nodal metrics of complex node representation, node information such as peer group ID and private ATM address, Internal and External reachable ATM addresses, horizontal link which includes a metric information element for each class of service (such as maximum cell rate, administrative weight, etc.), and about uplinks to higher level nodes. A significant change to any component of any TIG generates a significant change event for the containing PTSE. Such a significant change triggers an update to be sent by the corresponding node. Received updates are compared to the already available information. If the item received is found to be of a higher sequence number (local items with higher sequence numbers are exceptions and will be described later), it is inserted to the database and broadcast over all other links. Entries age and are eventually dropped unless they are renewed or updated. A special condition may occur when two previously disjointed network partitions become connected. In this case, the full topology database of each network needs to be broadcast over the other one. Consequently, when a link comes up, the full topology database is compared (through the exchange of sequence numbers) between any CPs that have become neighbors due to link recovery. New items that are received from the new neighbor are again inserted into the database and flooded to all neighbors except the one from which it was received.

Another special condition occurs when a node crashes and the last sequence number used by it is lost. To recover from such loss the following rules are applied. When the node comes up, it resets its sequence numbers for all local items. If its local entries that were distributed before the crash have all aged, then the reset has no impact on future operations. However, it might

be the case that some higher sequence values are stored at some other nodes. To overcome this problem, the node triggers a new and higher update every time it receives an update regarding one of its local items at a higher sequence number than the one it currently uses. A more complete description as well as proofs of correctness can be found in Section 7.0 and [3] [7].

## **6.2 Utilization update**

The utilization update is an iterative process triggered as a result of a major change in a dynamic database item, and conveys information which is crucial to the successful establishment or proper blocking of arriving calls. Clearly, the latency of this information update process reflects on the currency of the database and hence, the quality and promptness of the information used for call routing. Similarly, the maximum rate of the utilization updates that the system can process has a major impact on the granularity or precision of the information maintained in the nodes as well as on the network size (number of links and nodes) that can be supported in the nodal database. As the precision and number of links increase, so does the rate of utilization update that needs to be handled by each node. We use several mechanisms to facilitate and accelerate the process of utilization update. These improve both the promptness and scalability of the overall network control platform.

The underlying mechanism for the utilization update is the use of the native ATM multicast instead of a software-based flooding. The advantage is twofold. First, since the forwarding of ATM native multicast is performed at the switch (hardware) level, the software store-and-forward functions which would be involved in a flood are eliminated and the broadcast delay is largely reduced. The fact that forwarding is not necessary also allows batch processing (periodic polling of the utilization update queue) of these updates. Second, the native multicast provides each node with only a single copy of the utilization update message, compared to the flood process which delivers as many copies as there are incoming links attached to the node (CP cluster).

Utilization update is triggered whenever there has been a substantial change in the utilization from the values previously reported or if enough time has elapsed since the last update. The mechanism to determine what constitutes a significant change is part of the PNNI which, for utilization data, is defined by the percentage of reduction or increase of the available bandwidth over a link (with some minimum requirement as well, to avoid very small changes at low residual capacity). The rate of utilization update is expected to be orders of magnitude higher than the rate of the topology update. On the receiving side, the rate of utilization updates received can burden the receiving CP. This statement addresses the fact that handling each utilization update individually and instantaneously may result in too many system calls (interrupts) and hence an unacceptable nodal overhead. Periodic polling of the incoming cells queue (which is identified by a particular VC value) seems a better practice in terms of system performance. Note that utilization update corresponds to a particular TIG. Therefore, in order to maintain synchronization between the two mechanisms, the current corresponding sequence number is attached to the utilization update.

Utilization update is performed over the distribution tree structure described in Section 3.0. The elected leader of the peer group is responsible for setting up and maintaining the distribution tree. If the pivot-based approach is to be used, the peer group leader serves as the pivot. It uses the multicast setup mechanism in order to setup the distribution tree as described in Section 5.7, and the leader election algorithm described in Section 6.3 to maintain the tree, as described in the following. When a topological change that affects the connectivity of the distribution tree occurs (such as link or node failure), the leader will send a new tenure message (defined in PNNI and OSPF) to all other nodes within the peer group. Every node that receives this message will release the distribution tree and stop forwarding cells on this tree. Since the tenure message is sent using a reliable flooding mechanism, every node in the peer group will eventually receive this message and release the distribution tree. After the leader sent the tenure message, it will compute a new distribution tree and setup this distribution tree as described in Section 5.7.<sup>14</sup>

### **6.2.1 Utilization Update Cell Structure**

PNNI defines eight link state parameters to be advertised through the PNNI link state updates. These are:

- Maximum Cell Transfer Delay (MCTD)
- Maximum Cell Delay Variation (MCDV)
- Maximum Cell Loss Ratio for cell loss priority 0 (MCLR0)
- Maximum Cell Rate (MCR)
- Administrative Weight (AW)
- Cell Rate Margin (CRM)
- Variance Factor (VF)
- Available Cell Rate (ACR)

While PNNI uses a reliable flooding procedure to disseminate the information OPENET normally suppresses this update in the case of a change in the dynamic parameters (to be defined below), and creates a utilization update instead.

The link state parameters MCTD, MCDV, and MCLR0 are fixed parameters, i.e., their values remain constant irrespective of whether new connections are added or existing connections are released at this node. The most frequently changing parameter is the ACR, as its value is affected by new connections being added or existing connections being released. Relative to this parameter, the MCR and AW parameters can be considered as static parameters. The ACR parameter along with the CRM and VF parameters, are used in the call admission control pro-

---

14. Another option can be used in order to enhance the setup of the distribution tree. According to this option, the leader will “patch” the disconnected sub-tree to the existing distribution tree. This may be efficient if the sub-tree consists of a small portion of the distribution tree, particularly a single leaf node. Nevertheless, recomputing the entire distribution tree may result in a more desirable one than the “patched” distribution tree.

cedure. While the CRM and VF values can change with new connections being added or existing connections being released, they are expected to change slower than the ACR (in particular, the VF parameter).

This leads to the conclusion that the only parameters that need to be updated frequently are the ACR, CRM and VF parameters. As these parameters are used in the generic connection admission control to determine whether a new connection can be accepted or not, updated values of these parameters are crucial to increasing the throughput of the setup process. We choose to include these parameters in the utilization update cells for the CBR, R-VBR and NR-VBR traffic classes. However, as will be described later, other parameters can be included in the utilization update cells instead of or in addition to these parameters. We use the ATM adaptation layer 3/4 cell format. This allows for the multiplexing of multiple messages on our distribution tree structure, and hence allows us to provide this structure as a service for higher layer applications. However, our utilization update messages consists of a single cell only. More details on the exact format and contents of these messages appear in [17].

### **6.3 Leader election algorithm**

A single real physical system is selected to perform the logical peer group functions of PNNI and some other OPENET functions, such as the manager of the distribution tree. This system is called a peer group leader (PGL) in PNNI. Here we give a high level description of an algorithm to select a peer group leader. This algorithm consumes fewer messages than the original PNNI leader election algorithm. The algorithm guarantees both an eventual election of a single leader in the connected network component as well as an indication to the leader that all the current network nodes have accepted this information (election results).

#### **6.3.1 High level description of the algorithm**

The algorithm is based on the Propagation of Information with Feedback (PIF) algorithm described in [15]. In this algorithm, the propagation of messages occurs in two waves:

- The first wave, from the node that starts the PIF into the network, uses flooding for purposes of propagating information; and
- The second, from the network back to the originating node (for purpose of acknowledgment) uses a tree marked during the first wave.

Assume that topological changes have stabilized and the topology databases are consistent (each node waits for a sufficient time after a topological change). Each node with the highest priority according to its local database (the node ID is used as a tie-breaker) will start the PIF algorithm by sending a PTSP which contains a new type of nodal information group (type TBD0) to all of its neighbors. (The TBD0 message is similar to the PNNI nodal information group type 97.) Note that at the same time, another node may have started its own PIF algorithm. Suppose a node,  $i$ , receives a PIF PTSP originated at a node,  $S_j$ , from its neighbor  $l$ . Node  $i$  first checks its own current leadership priority (the highest known priority in its local database). If the received packet contains a lower leadership priority, it discards this packet,

and hence prevents the PIF associated with this packet from successfully terminating at the originating node. Otherwise:

- node  $i$  marks the port from which the packet was received;
- if this is the first PIF PTSP originated at node  $S_l$  that  $i$  has received from  $l$ , node  $i$  denotes this neighbor with a special mark  $P(S_l, i)$ , and sends the packet to all neighbors except to  $P(S_l, i)$ .

The second phase begins once a node has observed that it has marked all of its incoming ports. (Such a node must exist, and in fact is a leaf of the tree formed by the links connecting each node to its special node.) Once this occurs, it sends a PTSE acknowledgment packet to its special node. When node  $i$  observes that it has received the corresponding PTSE acknowledgment packets from all neighbors, it sends a PTSE acknowledgment packet to  $P(S_l, i)$ . If a link failure was detected by node  $i$  while waiting to receive acknowledgment packets from all of its neighbors, it marks this link as if an acknowledgment packet has been received on it, but uses another (new) type of PTSE acknowledgment packet (type TBD1). If node  $i$  receives a type TBD1 PTSE acknowledgment packet from any of its neighbors, then when it sends an acknowledgment packet to  $P(S_l, i)$ , it sends type TBD1 PTSE acknowledgment packet.

If node  $S_l$  receives acknowledgment packets from all neighbors, it can interpret this as a signal that its packet has indeed reached all connected nodes, and hence, all connected nodes know that node  $S_l$  is the leader (see proof in [15]). Notice however that the leader still needs to send a tenure message for all other nodes to determine that the leader election procedure has ended. If node  $S_l$  receives type TBD1 PTSE acknowledgment packets on any of its ports, it needs to start a new election. If  $K$  nodes start a PIF, then the number of packets sent is at most  $2K$  times the number of links in the peer group ( $K$  is typically 1). This is much smaller than the number of messages required in the PNNI leader election algorithm which is equal to two times the number of nodes in a peer group, times the number of links in the peer group.

## 6.4 Route computation

Route computation is needed during the setup process of new connections or the rerouting of existing connections. It is performed at the source CP of a connection or by another CP for third party setup. Route computation is required in order to obtain a simple path (in the unicast case) or tree (in the multicast case) in the network which meets the bandwidth and other constraints (such as delay and loss) of a connection. In addition, it is desirable to minimize the cost (administrative weight) of the computed route. According to PNNI, the link state parameters are classified into two classes, link metrics and link attributes. A link metric is a link state parameter that requires the values of the parameter for all links along a given path to be combined to determine whether the path is acceptable for carrying a given connection. A link attribute is a link state parameter that is considered individually to determine whether a given link is acceptable for carrying a given connection. Currently, the MCTD and the MCDV are the only link state parameters defined as link metrics which is combined by addition along the path. All other parameters are defined as link attributes. In the following, we describe the route computation process for the unicast and multicast connections.

### **6.4.1 Route computation for unicast connections**

A setup request for a unicast connection contains the bandwidth requirements and QoS class of the connection in both directions. The bandwidth requirements (i.e., the peak and sustainable cell rates) along with the current link state parameters (i.e., the ACR, CRM and VF) are used by the GCAC algorithm to determine whether a given link can be considered in the route computation. The QoS class determines the values of the MCTD, MCDV and MCLR0 parameters. The MCDV and MCLR0 parameters of a given link determine whether this link can be considered in the route computation. The route computation then proceeds by finding the minimum weight (according to the AW parameter) path under the constraint that the sum of the MCTDs along the path doesn't exceed the requested MCTD of the connection. A fast setup process is extremely important for the success of ATM networks as discussed in Section 1.1. An on-line computation of the route upon the receipt of a setup request can extensively delay the setup process (even for the non-constrained route computation). Hence, our route computation mechanism accounts for an off-line computation of routes, where the routes for different QoS classes and route capacities are pre-computed and stored in a route database, as described below. This allows for background computation of routes, where idle processor periods can be utilized by the route computation process. There is, of course, a memory overhead associated with this process for the storage of the routes. Off line route calculation is optimized for a network where the number of calls is high and each call request a small amount of bandwidth compared to typical link capacities. (In a well designed network, the number of calls requesting a large portion of the capacity should not be too large.) If this is the case, then it can be assumed that cached routes can be used multiple times, as single calls do not load them immediately. It might be reasonable to use both precomputed routes for frequent small bandwidth calls as well as on demand computation for large bandwidth calls.

Another aspect of the routing algorithm is the way it adapts to changes of topology and utilization. OPENET has taken the approach that a single instance of route computation will not be interrupted in the middle due to new changes. Instead, the current computation (of the specific class and bandwidth) will be completed and only then will the new topology parameters will be integrated. The computation will continue in the same round robin order until all cases are analyzed with the new parameters. In general, routes for high bandwidth calls are always computed first (for all QoS classes, as they can also route low bandwidth calls), and routes for low bandwidth calls are computed later. The latter approach guarantees that even in a frequently changing network, the routing algorithm will be able to make progress and all QoS will be served by it. If computations are interrupted or the algorithm always start again at the same point, livelocks may occur.

In the following we describe the route computation process and the resulting route data structures for the CBR QoS class. A similar computation is performed for the NR-VBR and R-VBR QoS classes. We first describe the algorithm for unidirectional connections. Our algorithm is a heuristic one based on several applications of Dijkstra's shortest path algorithm each with a different set of edge weights. Each application looks for an optimal set of paths with respect to one criterion using the others as constraints. The execution of the complete algorithm results in a set of feasible routes in terms of the delay requirements. This set of routes is stored as an ordered set of subtrees (which is a natural outcome of Dijkstra's algorithm). We distinguish

between small connections (e.g., 0-3 Mbps), medium connections (e.g., 3-15 Mbps) and large connections (e.g., 15-50 Mbps) and consequently execute the algorithm for each of these sizes.<sup>15</sup>

Let  $D^{CBR}$  denote the delay constraint of CBR connections and let  $w_i$  and  $D_i$  denote the weight and delay of link  $i$ , respectively. Let  $\bar{w}$  denote a vector whose components are  $w_i$ . Similarly denote  $\bar{D}$ . Denote by  $DIJ(\bar{x})$  the application of Dijkstra's algorithm to our network when the link weights equal  $\bar{x}$ ; Let  $DIST(T, \bar{x})$  be a vector whose  $i$ -th component is the distance of node  $i$  from the root of tree  $T$  when  $\bar{x}$  is taken as the link-weight vector of  $T$ . The algorithm uses a scalar constant  $c > 1$  and proceeds as follows:

1. Compute  $T^1 = DIJ(\bar{w})$ ,  $\bar{w}^1 = DIST(T^1, \bar{w})$ ,  $\bar{D}^1 = DIST(T^1, \bar{D})$ . Remove from  $T^1$  all nodes  $i$  for which  $D_i^1 > D^{CBR}$ .
2. Compute  $T^0 = DIJ(\bar{D})$ ,  $\bar{w}^0 = DIST(T^0, \bar{w})$ ,  $\bar{D}^0 = DIST(T^0, \bar{D})$ . Remove from  $T^0$  all nodes  $i$  for which  $D_i^0 > D^{CBR}$  (those nodes cannot be assigned a path in this round of computation) or those for which  $w_i^0 > c \times w_i^1$ .
3. Define  $p) = \frac{p}{\text{Max?} \{w_i^0\}} \bar{w} + \frac{(1-p)}{\text{Max?} \{D_i^0\}}$
4. Perform the following for  $p = 0.7, 0.4, 0.1$ :
  - Compute  $T^p = DIJ(\bar{x}(p))$ ,  $\bar{w}^p = DIST(T^p, \bar{w})$ ,  $\bar{D}^p = DIST(T^p, \bar{D})$ . Remove from  $T^p$  all nodes  $i$  for which  $D_i^p > D^{CBR}$  or those for which  $w_i^p > c \times w_i^1$ .

If at any step all destinations have been reached or have been determined to be unreachable, then the algorithm stops. All destinations not included in any of the trees are considered non-reachable under the current constraints. For routing to destination node  $i$  chooses the first path (if any) found in the trees  $T^p$  as  $p$  varies from 1 down to 0.

It should be noted that a better algorithm would perform step 4 for (almost) continuous values of  $p$  at a higher computation cost. Different values can be chosen based on computation resource availability or a relaxation method can be used to find the optimal  $p$ .

For bidirectional connections where both directions use the same undirected path, we modify the above algorithm in the following manner. We perform Dijkstra with link weights equal to the sum of weights in both directions and in each step we check for the bandwidth feasibility of the links in both directions. In Step 2 we apply Dijkstra with link weights equal to the maximum between the delays in both directions. All paths which meet the delay constraints in both directions and their weight is less or equal to the corresponding weight of Step 1 are included.

---

15. The computation is done for the high value of each size range. However, for large connections for which the route does not exist we perform on-line route computation with the requested bandwidth.

---

For other destinations we proceed with link weights equal to the maximum between the weights, in both directions.

For ABR and UBR connections the route computation is less complex due to the fact that no additive constraints are required. In this case Dijkstra algorithm with links weights equal to the administrative weights will do (after trimming those links where the loss, rate and other constraints are not met).

#### **6.4.2 Route computation for multicast connections**

Based on the bandwidth and QoS requirements of the multicast connection a subgraph of the network which contains only those links which can support the requested requirements is constructed. An approximated minimum weight Steiner tree which contains the source and the destinations is then computed using a heuristic algorithm (see, e.g. [16]), with links weights equal to the sum of the weights in both directions. Then, the delay feasibility of the tree is checked, and if it is not feasible we construct a shortest path tree from the source to the destinations using Dijkstra algorithm with the delays as weights and check for the delay feasibility of this tree. If it is not feasible then the route computation fails. Otherwise, we compare the weight of the tree with the Steiner tree weight. If it exceeds it by  $c$  times the route computation fails, otherwise it succeeds.

## **7.0 PNNI Functions Used in OPENET**

---

The OPENET approach is to use as many PNNI functions as possible without compromising the efficiency and speed of network control. This facilitates the OPENET complete interoperability with PNNI as the ATM internetworking standard. For example, PNNI specifies a hierarchy of Peer groups, which form administrative or functional domains. OPENET supports this hierarchy. PNNI defines at the Hello protocol which OPENET fully supports. Details on these can be found in [17] and [3].

## **8.0 References**

---

- [1] ATM Forum, "ATM User-Network Interface Specification," version 3.0, Prentice Hall 1993.
- [2] R. Handel, M. Huber and S. Schroder, *ATM Networks, Concepts, Protocols, Applications*, second edition, Addison-Wesley, 1994.
- [3] PNNI SWG, "PNNI Draft Specification" *ATM Forum* contribution 94-0471R7.
- [4] R. Cohen, B. Patel, F. Schaffa and M. Wiilebeek-LeMair, "The Sink Tree Paradigm: Connectionless Traffic Support on ATM LANs," *Proceedings of IEEE INFOCOM'94*, pp. 821-828, Toronto, Canada, June 1994.

---

## References

---

- [5] I. Cidon, I. Gopal, M. Kaplan and S. Kutten, "A Distributed Control Architecture of High-Speed Networks," *IEEE Trans. on Communications*, Vol. 43, No. 5, pp. 1950-1960, May 1995.
- [6] B. Awerbuch, I. Cidon and S. Kutten, "Communication-Optimal Maintenance of Replicated Information," *Proceedings of 31st Ann. Symposium on Foundation of Computer Science* (St. Louis, MO), pp. 492-502, October 1990.
- [7] D. Bertsekas and R. Gallager, *Data Networks*, second edition, Prentice Hall, 1992.
- [8] I. Cidon and I.S. Gopal, "PARIS: An Approach to Integrated High-Speed Private Networks," *International Journal of Digital and Analog Cabled Systems*, Vol. 1, No. 2, pp. 77-86, April-June 1988.
- [9] J. Moy, "OSPF Version 2," Network Working Group, Internet Draft, September 1993
- [10] I. Cidon, I. Gopal and R. Guerin, "Bandwidth Management and Congestion Control in plaNET," *IEEE Communication Magazine*, Vol. 29, No. 10, pp. 54-63, October 1991.
- [11] I. Cidon, I. Gopal, P. Gopal, R. Guerin, J. Janniello and M. Kaplan, "The plaNET/ORBIT High Speed Network," *Journal of High Speed Networks*, Vol. 2, No.3, pp. 171-208, 1993
- [12] A. Baratz, J. Gray, P. Green, J. Jaffe and D. Pozefsk, "SNA Networks of Small Systems," *IEEE Journal on Selected Areas in Communications*, Vol. SAC-3, No. 3, pp.416-426, May 1985
- [13] S. Even, *Graph Algorithms*, Computer Science Press, 1979
- [14] J. McQuillan, I. Richer and E. Rosen, "The New Routing Algorithm for the Arpanet," *IEEE Trans. on Communications*, Vol. 18, No. 5, pp. 711-719, May 1980.
- [15] A. Segall, "Distributed Network Protocols," *IEEE Trans. on Information Theory*, Vol. IT-29, No. 1, January 1983.
- [16] F. K. Hwang and D. S. Richards, "Steiner Tree Problems," *Networks*, Vol. 22, pp. 55-89, 1992.
- [17] I. Cidon, T. Hsiao, P. Jujjavarapu, A. Khamisy. A. Parekh, R. Rom and M. Sidi, "OPENET Architecture Specification," SUN Microsystems Inc., Version 1.1, May 1995.