Connection Management Software System (CMSS) Architecture

John D. DeHart

Applied Research Laboratory
Department of Computer Science
Washington University
St. Louis, Missouri 63130
jdd@arl.wustl.edu

Working Note ARL-95-03
Version 1.2
Original 1995
Revised January 1998
Revised June 1998

Abstract

In this document we describe the architecture for our Connection Management Software System. This software system has been under development in the Applied Research Laboratory of the Computer Science Department at Washington University since 1989. The architecture and the design of individual components have gone through a great deal of evolution over the years. The current state of the system is described. The development of this object oriented software system is being done in C++.

1. Introduction

In this document we describe the architecture for our Connection Management Software System. This software system has been under development in the Applied Research Laboratory of the Computer Science Department at Washington University since 1989. The architecture and design of individual components have gone through a great deal of evolution over the years. A lot of the high level design of the architecture was done in collaboration with Bellcore and Southwestern Bell. The development of this object oriented software system is being done in C++.

There are several other documents that describe individual components of the CMSS in more detail. All of these

Topic	Report Number	Author	Title
CMSS	ARL-95-03	DeHart	Connection Management Software System (CMSS) Architecture
CMAP	WUCS-94-21	DeHart and Cox	Connection Management Access Protocol (CMAP) Specification - Version 3.0
CMNP	ARL-94-14	Wu and DeHart	Connection Management Network Protocol (CMNP) Specification-Version 1.0 DRAFT
GNBSC	ARL-94-12	Wu, Cox and DeHart	GBNSC: The GigaBit Network Switch Controller
Jammer	ARL-96-01	Beal	Jammer Language Description: A Script Language for GigaBit Switch Testing
NCCP/NCMO	ARL-96-03	Wu and DeHart	Node Controller Management Object (NCMO) and Node Controller Communication Protocol (NCCP)
Installation	ARL-96-02	DeHart	Washington University GigaBit Network Software Installation and Start-up - DRAFT

Table 1. Documentation Roadmap

documents can be accessed through the Applied Research Laboratory WWW pages which can be found at:

http://www.arl.wustl.edu/arl

2. The Connection Management Software System

2.1 Software Architecture Overview

We anticipate that the heterogeneity of ATM switching network equipment will add greatly to the complexity of controlling these networks. As shown in Figure 1, networks will consist of heterogeneous switching systems produced by various vendors which may differ considerably in their control requirements and protocols, although all will presumably conform to the appropriate ATM standards. The same will undoubtedly be true of the client terminals connected to the network. A third source of network heterogeneity lies in the links connecting the switches and terminals, which will vary in bandwidth.

Managing such networks requires the introduction of a number of control abstractions which serve to encapsulate and conceal the differences in equipment. Figure 2 illustrates a number of these abstractions. A *node* abstracts a switch or collection of switches, providing a view of the group as a single large switch with a known interface and capabilities. A *control processor* (CP) is an abstraction of the control software for a single node; in some cases the software may actually run on a single machine, while in others it may be distributed. In our control model, this software is the Core Connection Management Software System (CMSS). The control software for multiple nodes must communicate to set up inter-nodal connections. This communication is in accord with specified, uniform *Network Node Interface (NNI) protocols* (*e.g.*, CMNP[13] in our system). Finally, the terminal-network control interface is encapsulated by *User to Network Interface (UNI) protocols* which specify the manner in which clients request connections through the network. Examples of such UNI protocols would be CMAP[11] and Q.2931 (or Q.93B) [1, 2, 8].

The Core CMSS present at each node is structured in three layers as shown in Figure 3. The *Connection Management Layer* is distributed across all the nodes of the network, and uses the NNI protocols of Figure 2 to set up inter-nodal connections. At each node, the Connection Management Layer communicates with the *Node Management*

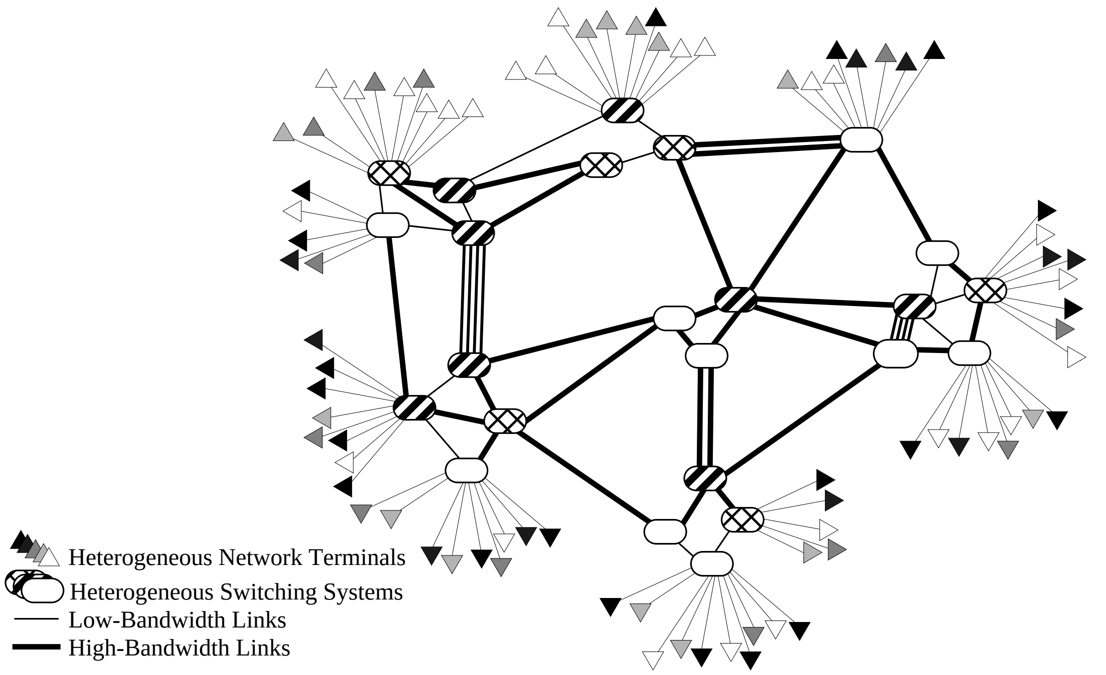
Heterogeneous Network Terminals
Heterogeneous Switching Systems
Low-Bandwidth Links
High-Bandwidth Links

Figure 1. Heterogeneous ATM Networks.

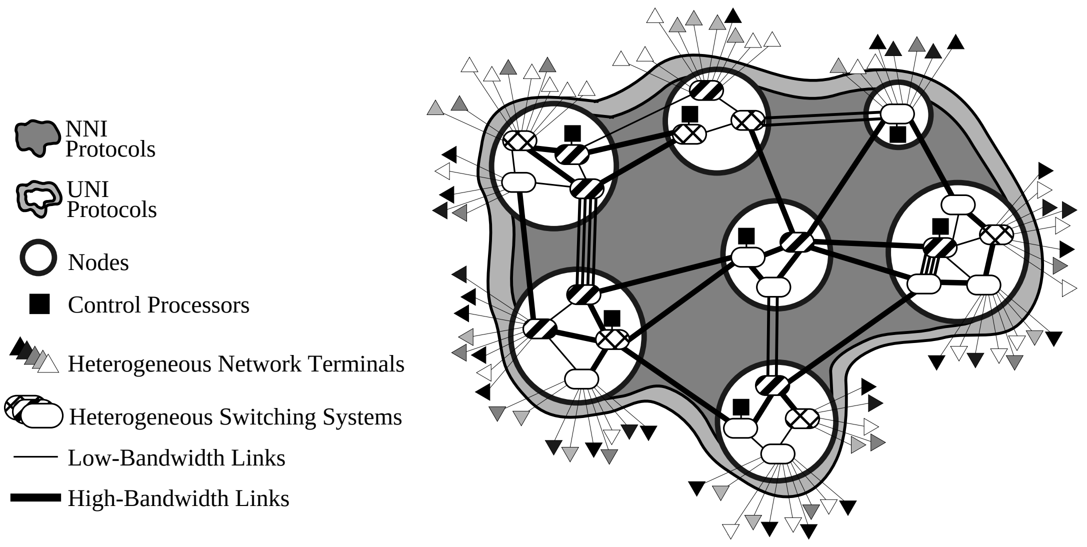

Figure 2. Control Abstractions for ATM Networks.

Layer for that node. The Node Management Layer abstracts the collection of switches in the node so that they are presented as a single large switch to the Connection Management Layer. The Node Management Layer is responsible for managing <u>intra</u>-nodal connections within the node. It issues commands to the *Switch Management Layer*, which handles connections within the individual switches and conceals hardware dependencies from the other layers. The UNI protocols are implemented by a fourth layer, the *Session Management Layer*, which is not considered part of the core CMSS. This layer is only necessary at nodes which support links to clients. It abstracts the whole network and presents a unified view of it to the clients.

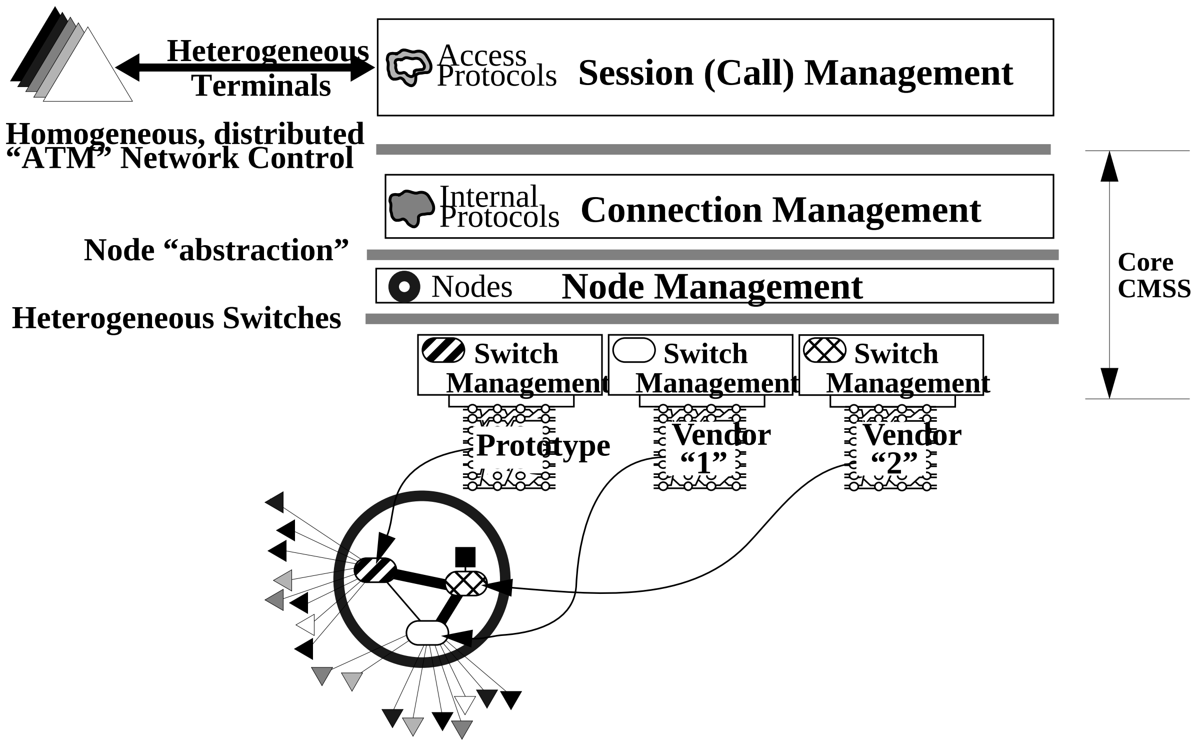

Figure 3. Core Connection Management Software System Plus Session Management

In the current design, the Core CMSS is realized as a tree of processes running on the node's CP, as shown in Figure 4. The top process in the tree is the *Connection Manager* (CM) for the node. This process communicates with the CMs of other nodes and with one subsidiary process. For the node shown in the figure, where three switches are grouped and managed as a single node, the subsidiary process is a *Node Controller* (NC). This NC in turn communicates with its subsidiaries; in the example these are one *Switch Controller* (SC) for each switch. If, as in the example, the switch hardware is supplied by several different vendors, these SC processes will be from different executables, each tailored to control the specific hardware. However, the API provided by each SC is identical, and is also identical to the NC API.

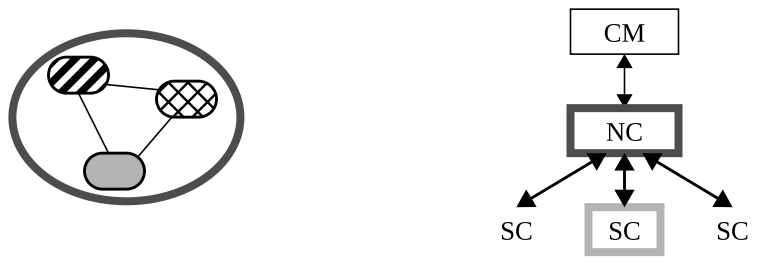

Figure 4. Example CMSS Process Structure.

CMSS processes communicate only along the links of the process tree. For example, in Figure 4, the CM and NC may communicate and the NC may communicate with each SC. However, the CM does not communicate directly with the SCs, nor do the SCs communicate with one another — indeed, the encapsulation provided by the CMSS is such that these processes have no knowledge that the others exist.

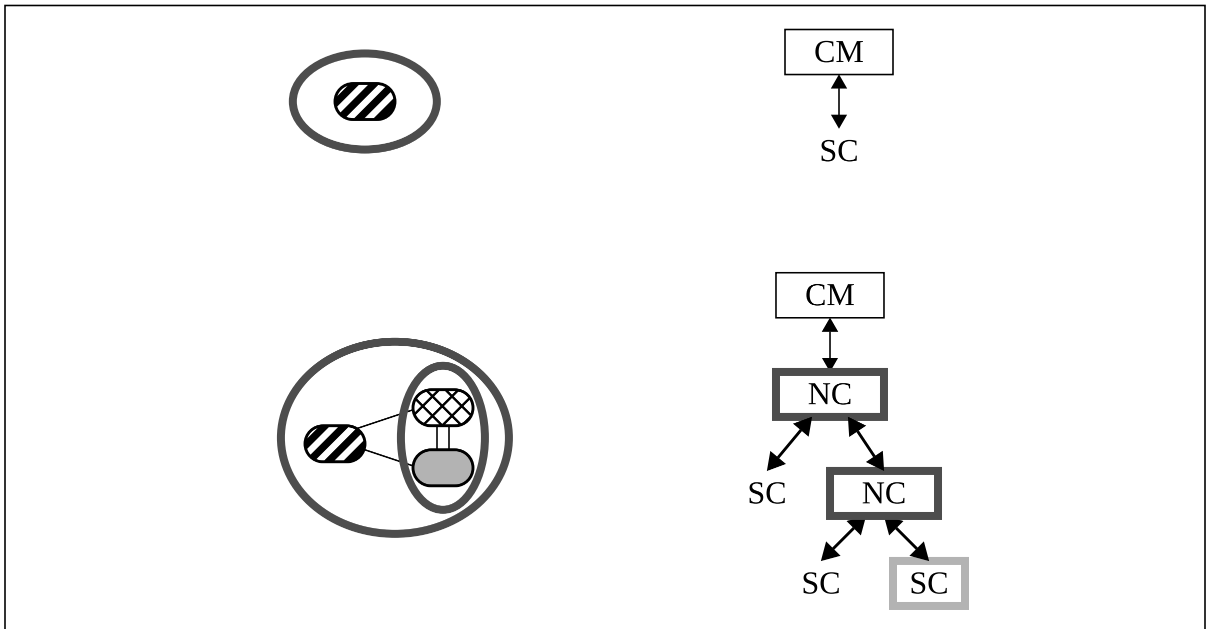

Figure 5. Additional CMSS Process Structures.

2.2 Interfaces Between Processes - The MO-API Model

The model used for the interfaces between processes in the CMSS is one of a Managed Object Application Programmers Interface. As shown in Figure 6, the development of Process Y includes the development of a MO-API to

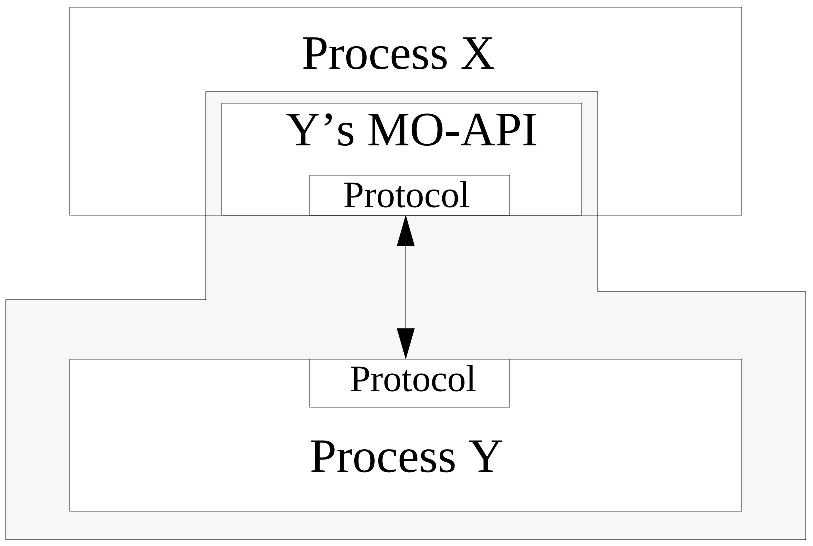

Figure 6. Managed Object Application Programmers Interface Model

be used by Process X and the protocol necessary to communicate between the MO-API and Process Y. When Process X creates the MO-API, it is the responsibility of the MO-API to take the necessary steps to spawn Process Y and set up the appropriate communications mechanism. Process X then manipulates the objects presented by the MO-API in order to use the services offered by Process Y. In fact, it is not, in general, necessary for Process X to understand that there is another process below the MO-API.

2.3 Common Code

Software processes in the CMSS are built using objects from the ARL/Project Zeus common code library. The objects in this library are primarily concerned with interprocess communication and context switching, and are constructed for a UNIX environment with shared memory and semaphores. The next several paragraphs provide a brief description of some of these objects.

ByteBuffer. A ByteBuffer is an array of bytes. Overloaded shift operators are supplied, allowing the user to store binary data in and retrieve binary data from the buffer. ByteBuffers are used primarily to format messages used in interprocess communication, to guarantee that the format and interpretation of the data is uniform — even if the two processes run on different machines, or were built with different compilers.

Scheduler. A Scheduler_class object provides a means for scheduling timeouts using a shared-memory timer. Objects that should time out are derived from the SCHEDULER_base_class, and must provide a virtual *timed_out()* function which is called when a timeout occurs. The timeout is scheduled by providing the derived object to a scheduler object's registration function. When the scheduler object's *touch()* function is called, the object examines its internal lists to determine if any timeouts have occurred. If so, it calls the *timed_out()* functions for those objects.

Rendezvous objects. The rendezvous mechanism provides a safe means to access objects, which is robust even if those objects have been deleted. Objects are derived from the REND_base_class and must provide a virtual *REQ_rendezvous()* function. A rendezvous "marker" may be extracted from such an object and treated as simple data — for example it can be transmitted to another process, which can then send the marker back. The marker can be used to request a rendezvous with the original object, supplying a pointer to (arbitrary) data. If the object still exists, its *REQ_rendezvous()* function will be called with the supplied data pointer. If the object does not exist, the rendezvous request is ignored.

Join objects. The JOIN base classes provides a standard mechanism for building parent-child and sibling relations between objects.These objects guarantee the safe deletion of child objects when the parent is deleted. Efficient management of addition, deletion and maintenance of the lists involved in a 1-many relationship is also provided.

TCPObject. The TCPObject class provides an easy to use wrapper around the system calls needed to set up and use TCP socket connections.

PRIQUE and PriquePair. A PRIQUE is a communications link between two processes on the same machine. The PRIQUE is implemented using shared memory and is suitable for message-based communications. Several varieties of PRIQUE are supported, including ones with high-priority (out-of-band) messages and forward stores (messages stored in local memory until there is room in the shared memory). The PriquePair class implements two PRIQUEs between a pair of processes and includes a small protocol supplying send and receive functions. Each process sends on one PRIQUE and receives on the other.

Context_switcher. A Context_switcher object accesses a shared memory segment. Processes using the Context_switcher register their PRIQUEs, file descriptors and other information in the shared segment. When a process determines that it may not have any additional work (because, *e.g.*, it is waiting for input) it calls the *Switch_If_Idle()* function of the object. This function decides if the process should be swapped out, examining the shared-memory PRIQUEs to determine if any input may be available. If the Context_switcher decides the process should be swapped, it selects another process that is registered with the Context_switcher and sends a UNIX signal to that process. This causes the original process to be swapped out, and the other process to be scheduled for swapping in, by the operating system.

The Context_switcher object was of great use with SunOS4.x. We no longer use most of it. The only thing it is still in use for is its interface to select().

ATMCard. The interface to the ATM link is hardware-dependent. The ATMCard object provides a uniform API for this link.

NCCP. The Node Controller Communications Protocol [28] defines the interface between two processes in the CMSS tree, or between Jammer (Section 3) and a GigaBit Network Switch Controller [27]. The protocol is message-based and uses many of the objects described above.

NCMO. The Node Controller Managed Object [28] defines the API that can be used to build general multipoint connections in the switch. Operations on NCMO objects cause NCCP messages to be sent to the switch controller which in turn manipulates connections in the switch.

3. Jammer

Jammer [4] is a laboratory testing tool that we have developed to aid in the testing of the prototype switches that we develop. It provides a programming language interface with block and conditional constructs like procedures, while loops and if-then-else blocks as well as commands to directly manipulate the hardware tables in a particular prototype switch. This provides the test engineer with a very powerful mechanism for generating automated tests to verify the correct operation of the switch.

Jammer communicates directly to a switch controller via the NCCP (Figure 7).

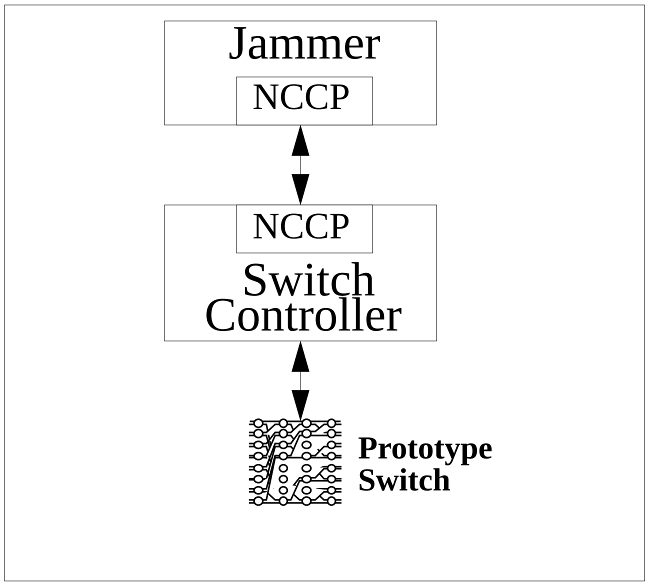

Figure 7. Jammer

4. Conclusion

We have given a general overview of the software that comprises the Connection Management Software System. Separate documents describe some of the components in more detail. Anyone wanting to utilize or modify any of those components should refer to the specific documents covering them and to the source code.

References

[1] ANSI T1S1 Technical Sub-Committee. Broadband Aspects of ISDN Baseline Document. T1S1.5/90-001, June 1990.

[2] ATM Forum, “The ATM Forum Technical Committee User-Network Interface (UNI) Specification Version 3.1”, The ATM Forum 1994.

[3] ATM Forum, “ATM Forum 94-0471R7 PNNI Draft Specification”, The ATM Forum 1994.

[4] O.M. Beal, “Jammer Language Description: A Script Language for GigaBit Switch Testing,” Washington University, Applied Research Laboratory Working Note ARL-96-01, March 1996.

[5] R. G. Bubenik, J. D. DeHart and M. E. Gaddis. “Multipoint Connection Management in High Speed Networks.” In IEEE Infocom ‘91: Proceedings of the Tenth Annual Joint Conference of the IEEE Computer and Communications Societies, pages 59-68, April 1991.

[6] R. G. Bubenik, M. E. Gaddis and J. D. DeHart. “A Strategy for Layering IP over ATM”. Washington University Applied Research Laboratory, Working Note 91-01, Version 1.1, April 1991.

[7] R.G. Bubenik, M.E. Gaddis, and J.D. DeHart. “Virtual Paths and Virtual Channels.” In IEEE Infocom ‘92: Proceedings of the Eleventh Annual Joint Conference of the IEEE Computer and Communications Societies, May 1992.

[8] CCITT. Recommendations Drafted by Working Party XVIII/8 (General B-ISDN Aspects) to be Approved in 1992, Study Group XVIII—Report R 34, December 1991.

[9] CCITT Recommendation Q.931 (I.451), ISDN User-Network Interface Layer 3 Specification, Geneva, 1985.

[10] J. R. Cox and J. S. Turner. “Project Zeus Design and Application of Fast Packet Campus Networks”. Washington University, Department of Computer Science Technical Report 91-45, July 1991.

[11] K. Cox and J. DeHart. “Connection Management Access Protocol (CMAP) Specification,” Washington University, Department of Computer Science Technical Report WUCS-94-21, Version 3.0, July 1994.

[12] J. DeHart, “Washington University GigaBit Network Software Installation and Start-up”, Washington University, Applied Research Laboratory Working Note ARL-96.-02, DRAFT, March 1996

[13] J. DeHart and D. Wu, “Connection Management Network Protocol (CMNP) Specification,” Washington University, Applied Research Laboratory Working Note ARL-94-14, Version 1.0 DRAFT, September 1994.

[14] M. E. Gaddis, R.G. Bubenik, and J.D. DeHart. “Connection Management for a Prototype Fast Packet ATM B-ISDN Network.” In Proceedings of the National Communications Forum, vol. 44, pp. 601-608, October 8-10, 1990.

[15] M. E. Gaddis, R.G. Bubenik, and J.D. DeHart. “A Call Model for Multipoint Communications in Switched Networks.” submitted for publication to ICC ‘92, Chicago, Illinois, June 1992.

[16] S.E. Minzer. “Broadband ISDN and Asynchronous Transfer Mode (ATM).” In IEEE Communications Magazine, 27(9):17-24, September 1989.

[17] S.E. Minzer and D.R. Spears. “New Directions in Signaling for Broadband ISDN.” In IEEE Communications Magazine, 27(2):6-14, February 1989.

[18] G.M. Parulkar, J.S. Turner. Towards a Framework for High Speed Communication in a Heterogeneous Networking Environment. In IEEE Infocom ‘89: Proceedings of the Eighth Annual Joint Conference of the IEEE Computer and Communications Societies, pages 655-667, April 1989.

[19] G. M. Parulkar. “The Next Generation of Internetworking”. ACM SIGCOMM Computer Communications Review. vol. 20, no. 1, New York, NY, pp. 18-43, January, 1990.

[20] A.S. Tanenbaum. Computer Networks. Prentice-Hall, 1981.

[21] J. S. Turner, “Fast Packet Switching System”, U.S. Patent 4 494 230, January 15, 1985.

[22] J.S. Turner. “New Directions in Communications.” In IEEE Communications Magazine, 24(10):8-15, October 1986.

[23] J.S. Turner. “Design of an Integrated Services Packet Network.” In IEEE Transactions on Communications, 4(8):1373-1380, November 1986.

[24] J.S. Turner. “Design of a Broadcast Packet Switching Network.” In IEEE Transactions on Communications, 36(6):734-743, June 1988.

[25] J. S. Turner. “A Proposed Management and Congestion Control Scheme for Multicast ATM Networks.” Washington University, Computer and Communication Research Center Technical Report 91-01, May 1991.

[26] J.S. Turner, “A Gigabit Local ATM Testbed for Multimedia Application.” Washington University, Applied Re search Laboratory Technical Report ARL-94-11 Version 3.1, January 1996.

[27] D. Wu and J. DeHart, “GBNSC: The GigaBit Network Switch Controller,” Washington University, Applied Research Laboratory Working Note ARL-94-12, Version 1.2, June 1996.

[28] D. Wu and J. DeHart, “Node Controller Managed Object (NCMO) and Node Controller COmmunication Protocol (NCCP),” Washington University, Applied Research Laboratory Working Note ARL-96-03, June 1996.