

ADVANCED COMMUNICATIONS SYSTEMS

Jonathan S. Turner (PI)
September 1, 1986 – August 31, 1987

WUCS-87-22

Mark A. Franklin

Pierre Costa
Riccardo Melen

Shahid Akhtar
Neil Barrett
Victor Griswold
Mark Hunter
Shabbir Khakoo
George Robbert
James Sterbenz
Bernard Waxman
Einir Valdimarssen
David Wexelblat

Copy to:

Shahid Akhtar
Akira Arutaki
Neil Barrett
Pierre Costa
Jerome R. Cox, Jr.
Maurizio Dècina — Italtel
Mark A. Franklin
Victor Griswold
Neil Haller* - Bellcore
Joe Hui - Bellcore
Mark Hunter
Akihiro Kitamura* - NEC
Shabbir Khakoo
Tony Lee - Bellcore
Riccardo Melen
Eric Nussbaum — Bellcore
George Robbert
Anna Robrock* - Italtel
J. Shimuzu - NEC
Dave Sincoskie - Bellcore
Barry Spielman
James Sterbenz
N. Suzuki - NEC
Giuseppe Valbonesi - Italtel
Einir Valdimarssen
Bernard Waxman
Ken Wong
Liang Wu - Bellcore
T. Yamaguchi - NEC

Research Objectives

The Advanced Communications Systems Project is concerned with new communications technologies that can support a wide range of different communications applications in the context of large public networks. Communications networks in common use today have been tailored to specific applications and while they perform their assigned functions well, they are difficult to adapt to new uses. There currently are no general purpose networks, rather there are telephone networks, low-speed data networks and cable television networks. As new communications applications proliferate, it becomes clear that in the long term, a more flexible communications infrastructure will be needed. The Integrated Services Digital Network concept provides a first step in that direction. We are concerned with the next generation of systems that will ultimately succeed ISDN.

The main focus of the effort in the ACS project is a particular switching technology we call broadcast packet switching. The key attributes of this technology are (1) the ability to support connections of any data rate from a few bits per second to over 100 Mb/s, (2) the ability to support flexible multi-point connections suitable for entertainment video, LAN interconnection and voice/video teleconferencing, (3) the ability to efficiently support bursty information sources, (4) the ability to upgrade network performance incrementally as technology improves and (5) the separation of information transport functions from application-dependent functions so as to provide maximum flexibility for future services.

Acknowledgements

The Advanced Communications Systems Project operates within the Computer and Communications Research Center, an inter-departmental research laboratory in the School of Engineering and Applied Science at Washington University. The ACS project began on January 1, 1986 with support from Bell Communications Research and Italtel SIT. Additional funding is now provided by NEC and the National Science Foundation through grant DCI 8600947.

The Center's research program seeks an appropriate balance between theoretical and practical issues and has attracted considerable interest world-wide. Program sponsors interact with the Center through exchange of information and personnel. Our current sponsors are

National Science Foundation
Bell Communications Research
Italtel SIT
Nippon Electric Corporation

We thank all our sponsors for their collaboration and support. Special thanks go to Gil Devey and Steve Wolf at NSF, Eric Nussbaum and Neil Haller of Bell Communications Research, Maurizio Dècina and Anna Robrock of Italtel and Akihiro Kitamura and Takehiko Yamaguchi of NEC. We also thank Washington University for providing an excellent environment in which to carry out this work, in particular Dean James McKelvey and Jerry Cox for all their support and encouragement.

Contents

| | | |
|----------|--|-----------|
| 1 | Summary of Progress | 1 |
| 2 | Switch Architecture Studies | 13 |
| 2.1 | Comparison of Alternative Switch Fabrics | 13 |
| 2.2 | Refinements to the BPN Switch Fabric | 22 |
| 2.3 | Design of Large Systems | 24 |
| 3 | Performance Studies | 27 |
| 3.1 | Fluid Flow Loading Analysis | 27 |
| 3.2 | Generalized Non-Blocking Networks | 35 |
| 3.3 | Packet Misordering | 39 |
| 4 | Prototype Hardware Design | 41 |
| 4.1 | Packet Formats | 42 |
| 4.2 | Timing | 46 |
| 4.3 | Packet Switch Element | 47 |
| 4.4 | Packet Processor | 54 |
| 4.5 | Broadcast Translation Circuit | 60 |
| 5 | Tools for Design of Communication Circuits | 63 |
| 5.1 | Synchronous Streams Processors | 63 |
| 5.2 | Implementation of SSPs | 66 |
| 5.3 | Tools for Constructing Memories | 68 |
| 5.4 | Other Tools | 69 |

| | | |
|----------|---|------------|
| 6 | Connection Management | 71 |
| 6.1 | Specification of Multipoint Connections | 71 |
| 6.2 | Multipoint Control Protocols | 75 |
| 6.3 | Prototype Connection Management Software | 81 |
| 7 | Multipoint Routing | 83 |
| 7.1 | Approximation Algorithms | 83 |
| 7.2 | The Dynamic Steiner Tree Problem | 87 |
| 7.3 | Random Graphs and Probable Performance | 90 |
| 7.4 | Distributed Routing Algorithms | 90 |
| 8 | Bandwidth and Buffer Management | 93 |
| 8.1 | Queueing Behavior of Bursty Sources | 94 |
| 8.2 | Bandwidth Allocation | 96 |
| 8.3 | Bandwidth Specification and Enforcement | 98 |
| 8.4 | Multipoint Congestion Control | 99 |
| 8.5 | Access Arbitration in Multipoint Channels | 103 |
| 9 | Packet Video | 107 |

List of Figures

| | | |
|-----|--|----|
| 1.1 | Publications and Related Activities | 2 |
| 1.2 | Technical Reports | 3 |
| 1.3 | Graduate Student Staff | 11 |
| 2.1 | Broadcast Packet Switch Fabric | 14 |
| 2.2 | Starlite Switch Fabric | 16 |
| 2.3 | Alternative Starlite-Type Switch Fabric | 18 |
| 2.4 | Prelude Switch Fabric | 20 |
| 2.5 | Local Switch Design | 25 |
| 3.1 | Composition Operation | 29 |
| 3.2 | Recursive Construction of Delta Network | 30 |
| 3.3 | Construction of Delta Network with Distribution Stages | 32 |
| 3.4 | Construction of Alternate Routing Network | 33 |
| 3.5 | Worst-Case Configuration for Copy Network | 34 |
| 3.6 | Distribution of Delay and Misordering Probabilities | 39 |
| 4.1 | Prototype Switch Module | 42 |
| 4.2 | Packet Formats | 44 |
| 4.3 | Local and Global Timing Relationships | 47 |
| 4.4 | External Interface for Packet Switch Element Chip | 48 |
| 4.5 | Block Diagram of Packet Switch Element Chip | 51 |
| 4.6 | Input Circuit | 53 |

| | | |
|------|--|-----|
| 4.7 | Input Control Circuit | 55 |
| 4.8 | External Interface for Packet Processor | 56 |
| 4.9 | Packet Processor Circuit | 57 |
| 4.10 | External Interface for BTC | 60 |
| 4.11 | Block Diagram of Broadcast Translation Chip | 62 |
| 5.1 | Generic Synchronous Stream Processor | 64 |
| 5.2 | Target SSP Architecture | 66 |
| 5.3 | Structure of A Silicon Compiler | 67 |
| 5.4 | Packet Buffer | 68 |
| 5.5 | Lookup Table | 69 |
| 6.1 | One-to-Many Connection | 72 |
| 6.2 | Connection for Video Lecture | 74 |
| 6.3 | Connection Management Architecture | 81 |
| 7.1 | An Example of the Application of MST | 84 |
| 7.2 | Experimental Performance of MST and RS | 85 |
| 7.3 | Dynamic Greedy Algorithm with Sequence: a,b,d,f,e, \bar{f} , \bar{d} | 88 |
| 7.4 | Experimental Results for the Greedy Algorithm | 89 |
| 8.1 | Markov Chain Model | 95 |
| 8.2 | Packet Loss Rates | 96 |
| 8.3 | Effective Bandwidth | 97 |
| 8.4 | Simple Bandwidth Enforcement Mechanism | 98 |
| 8.5 | Buffer Management Mechanism | 101 |
| 8.6 | Buffer Implementation | 102 |
| 9.1 | Hybrid Video Codec | 108 |
| 9.2 | Fixed vs. Variable Rate Channels | 109 |

1. Summary of Progress

The research program of the ACS project can be divided into four major areas: (1) switching system architecture, (2) connection management, (3) network control problems, such as routing and congestion control and (4) design of communications applications in the context of broadcast packet networks. The primary focus of our effort in the last year has been switching system architecture, including the design of a prototype broadcast packet switch. In support of this prototyping effort we have also been developing tools to aid in the design of the custom integrated circuits to be used in the prototype. We have also made substantial progress on connection management, including the design and implementation of preliminary software, and we are continuing to make steady progress in the area of network control. Our work on application design is currently limited to an initial study of the issues associated with packetized video, focussing especially on the effect of packet transport on the design of video coding methods.

We have been active in publishing our results on broadcast packet switching. Papers have been presented at several conferences and revised versions have appeared or are scheduled to appear in leading journals. Patent applications have been filed on broadcast packet switching and invited lectures have been given at many industrial and academic laboratories. (See Figures 1.1,1.2 for details.) Our work has generated a great deal of interest throughout the world, and appears to be having an influence on the research programs at several major industrial laboratories. We find this impact of our work particularly gratifying and expect to see it continue as our research program develops.

The following subsections summarize the progress we have made in several specific areas during the past year and outline our plans for the coming year. More detailed accounts of each of these topics appear in later sections.

Switch Architecture and Hardware Design

The most novel aspect of our research program is its focus on networks supporting flexible multi-point communication. Any switching system supporting multi-point

Published Papers

"Design of a Broadcast Packet Switching Network," by Jonathan S. Turner, *Proceedings of Infocom 86*, pp. 667-675, 4/86. Also, to appear in *IEEE Transactions on Communications*.

"New Directions in Communications," by Jonathan S. Turner, *IEEE Communications Society Magazine*, 10/86.

"Design of an Integrated Services Packet Network," by Jonathan S. Turner, *IEEE Journal on Selected Areas in Communications*, 11/86.

"Performance of a Broadcast Packet Switch," by Richard Bubenik and Jonathan Turner. *Proceedings of ICC 87*, pp. 1118-1122, 3/86. Also, to appear in *IEEE Transactions on Communications*.

"The Challenge of Multipoint Communication," by Jonathan S. Turner, *Proceedings of the ITC Seminar on Traffic Engineering for ISDN Design and Planning*, 5/87.

Invited Lectures

Telenet Inc., Reston, VA (8/87)

Southwestern Bell Telephone, St. Louis, MO (8/87)

Bell Atlantic, Great Gorge, NJ (2/87)

Midwest Workshop on Communications Systems, St. Louis, MO (11/86)

Computer Communications Workshop, Warner Springs, CA (9/86)

ITT Advanced Technology Center, Shelton, CT (9/86)

Tutorial on "Integrated Networks for Diverse Applications," at *Infocom 87*.

Program committee for *ISS 87, ICC 87, Midwest Workshop on Communications Systems*.
Guest editor for special issue of *IEEE Journal on Selected Areas in Communications*

Course on switching systems (CS 577).

Figure 1.1: Publications and Related Activities

communication must be able to connect any subset of its incoming channels to any subset of its outgoing channels. This is in contrast to point-to-point switching systems which need only connect input-output pairs.

Our work is based on a particular switching system architecture for multipoint communication. During the past year we have also been studying some competing architectures, in particular the Starlite architecture based on Batcher's bitonic sorting network which is being developed at Bell Communications Research, and the Prelude system which is based on an extension of the classical time-slot interchanger design and is under development at CNET in France. Each architecture

“Performance of a Broadcast Packet Switch,” by Richard Bubenik and Jonathan Turner, WUCS-86-10.

“An Architecture for Connection Management in a Broadcast Packet Network,” by Kurt Haserodt and Jonathan Turner, WUCS-87-3.

“System Testing of a Broadcast Packet Switch,” by Shabbir Khakoo and Jonathan Turner, WUCS-87-4.

“Specification of Integrated Circuits for a Broadcast Packet Network,” by Jonathan Turner, WUCS-87-5.

“The Challenge of Multipoint Communication,” by Jonathan Turner, WUCS-87-6.

“Design of a Broadcast Translation Chip,” by George Robbert, WUCS-87-9.

“Thesis Proposal: Routing of Multipoint Connections,” by Bernard Waxman, WUCS-87-10.

“Fluid Flow Loading Analysis of Packet Switching Networks,” by Jonathan Turner, WUCS-87-16.

“Distributed Protocols for Access Arbitration in Tree Structured Communication Channels,” by Riccardo Melen and Jonathan Turner, WUCS-87-17.

Figure 1.2: Technical Reports

has its respective advantages and disadvantages and none clearly dominates the others. Prelude and Starlite have two properties not shared by our architecture; they preserve packet sequence and they allow a single shared buffer. The ability to preserve packet sequence is potentially important for certain high speed applications. Shared buffering can provide substantially lower packet loss rates in the presence of highly bursty traffic. Some initial studies of these issues have already been made and they will be explored in detail in the coming year, first to assess their real importance, and second to identify extensions of the basic switch architecture that can address them.

Work on a laboratory prototype of our switching system was started about sixteen months ago, when we began design work on two integrated circuit chips. The first of the two chips is the packet switch element that makes up the copy, distribution and routing networks. This is a multi-function switch element that can be configured for any of the three networks, with two input and output ports per switch element. The second chip is the broadcast translation circuit which performs the translation for multi-point packets. It contains two random access memories implementing a pair of lookup tables controlling the translation process, plus associated control circuitry. The chips are being designed in a scalable CMOS

process with two layers of metal.

These preliminary designs have just recently been completed and submitted for fabrication. We are using our experience with these preliminary designs to guide us in the specification and design of the next set of chips that we will incorporate in our laboratory prototype. We also plan to design a two chip implementation of a packet processor and possibly a datagram router. These are being tackled in a broader context. We have found that several of the chips we need contain similar parts which are profitably viewed as special cases of a more general *synchronous streams processor*. We are developing a special-purpose silicon compiler that will take as input a specification of a streams processor and produce a description of a circuit implementing that specification. Substantial progress has been made on this program in the past year; the program is currently capable of producing simple circuits and we are confident that its use will significantly reduce the effort required for the design of several of the chips we require. It will also provide a powerful tool for the design of other similar chips. We have also recently begun related efforts which seek to automate certain portions of the design process for our prototype system.

Performance of Packet Switching Fabrics

During the past year we have sought to extend our understanding of the performance of packet switching fabrics in general, with of course a special focus on the broadcast packet switch. One important result has been the development of a systematic method for analyzing the effect of different traffic patterns on the loading of internal links within a packet switching fabric. This method allows us to make statements about the worst-case loading of a variety of different switch fabrics. This has led to several new results quantifying the effect of distribution stages on switch fabric performance. One result shows that a k stage routing network requires an additional $k - 1$ distribution stages in order to avoid overloading of internal links. Another shows that just two distribution stages dramatically improve the worst-case performance of copy networks. Other results concern the effect of the number of ports per node on worst-case loading. Of special interest is the observation that the worst-case performance of copy networks deteriorates as the number of ports per node increases.

We have recently been seeking to generalize the classical theory of non-blocking networks to networks in which internal links can multiplex multiple connections, with each connection consuming an arbitrary fraction of the link's capacity (subject of course to the constraint that the sum of the connection loads is no more than the link's capacity). This is relevant to the design of large switching systems constructed from multiple switch modules. It is also important for switch fabrics

which route all packets of a given connection along the same path, such as the systems under development at CSELT and Bell Telephone Manufacturing. Our initial results include an analysis of the amount of expansion required to make Clos and Cantor networks strictly non-blocking. We are also investigating the amount of expansion needed to obtain a rearrangeably non-blocking system and studying methods of analyzing blocking probability in such networks.

We have also made some initial simulation studies quantifying the likelihood of packets getting out of sequence when passing through a broadcast packet switch fabric. The results from these and planned further studies will be used to help in the design of mechanisms to recover proper sequencing.

Connection Management

Connection management refers to the collection of algorithms used to create and maintain multi-point connections in a broadcast packet network. A multi-point connection is intended to be a flexible mechanism that can support a wide variety of different applications. To achieve this flexibility, it must be possible to configure a multi-point connection for different uses. One of the first challenges in creating a useful and practical connection management system is deciding exactly what set of primitive capabilities the network should provide to enable users to configure connections. The subsequent challenge is to design the mechanisms needed to implement these capabilities.

We have identified and refined a method of configuring connections based on the concepts of sub-channels within a connection and permissions. Sub-channels allow a connection to be broken down into several distinct information flows, which can be configured differently but because of their close relationships are controlled by the network in a unified way. Permissions give the user a mechanism for controlling access to sub-channels and assist the network in managing its resources (primarily trunk bandwidth).

Based on these ideas, we have developed a specification of a simple connection management architecture and a series of scenarios showing how it can be used to support a variety of applications including broadcast video and multi-media conferencing. The connection management architecture has been designed at several levels of abstraction, with explicit interfaces at each level. The primary abstraction level, from the user's perspective, is the one that defines the interface between the network and the user's *termination controller*. At this level, the network is viewed as a single entity which modifies connections in response to control messages. The next level of abstraction below this defines the interfaces between switching systems in the network and it is at this point that explicit reference

must be made to the distributed algorithms and data structures that implement the higher level abstractions. We have also been considering a higher level of abstraction corresponding to the interface between termination controllers. At this level application-dependent issues appear. Termination controllers cooperatively determine how connections should be configured to suit the client applications, and direct the network to configure them via control messages.

In the past year, we have developed an initial set of protocols supporting multi-point connections and implemented those protocols in the form of a software simulation that allows us to configure an arbitrary network, then set up and modify multi-point connections in that network. Our implementation of multi-point connections includes a general transaction mechanism for sequencing concurrent changes to a connection. We plan to use our current simulation to obtain a better understanding of the strengths and limitations of our current collection of protocols, including the transaction mechanisms on which they are based.

Routing

The objective of the routing problem is to determine a set of network resources (primarily trunk bandwidth) sufficient to support communication among a specified set of users. In conventional circuit switched networks, all connections require the same amount of bandwidth and (almost all) have exactly two endpoints. Such a network can be described formally as a graph in which each edge has both a capacity and a length. A set of connections for such a network is simply a collection of vertex pairs. A feasible route assignment is an assignment of each connection to a path joining the connection's endpoints that doesn't exceed the capacity of any edge. An optimum routing algorithm is one that can find a feasible assignment whenever one exists.

Of course, this version of the problem is a static one. In a real communications network, the set of connections changes with time and the network must implement a routing policy that manages the changing set of connections in a way that makes it unlikely that a new connection will be blocked. In the interests of efficiency, it is generally assumed that once a connection has been assigned a route, that assignment will remain fixed as long as the connection is present. These considerations lead to a routing policy based on the heuristic strategy of routing connections by the shortest path available at the time the connection is established.

If connections can have an arbitrary bandwidth associated with them, the routing problem becomes a bit more complicated. One must now consider the network to be a graph in which vertices can be joined by multiple edges. To prevent blocking of connections with large bandwidth requirements, new connections should be

assigned to the fullest edges with sufficient capacity along the assigned route. This strategy preserves large blocks of bandwidth for use by high speed connections.

In broadcast networks, a connection can involve an arbitrary number of endpoints. A feasible route assignment for a set of connections is an assignment of each connection to a subtree connecting its endpoints, in a way that does not exceed the capacity of any edge. As in the case of point-to-point networks, connections come and go over time, and so the appropriate routing policy is to assign each connection to the subtree with shortest total length available at the time the connection is established. This can be viewed as a generalization of the Steiner tree problem in graphs. This problem is known to be NP-complete, meaning that there is unlikely to be an efficient algorithm that can always find an optimal solution. On the other hand, there are several efficient algorithms that yield solutions that are close to optimal. The best known one is called the minimum spanning tree heuristic (MST).

Connections in broadcast networks are dynamic in another way. They grow and shrink with time as individual endpoints come and go. The challenge here, is to maintain a good connection topology without doing a great deal of recomputation each time an endpoint is added or dropped. Practical algorithms must be suitable for distributed implementation, with each node making decisions based on local information. The simplest algorithm is a greedy strategy that adds new endpoints by joining them to the connection by the shortest available path and dropping branches of the connection tree when endpoints drop out.

Our research objective is to develop practical and efficient algorithms that can be used in actual multi-point communication networks. To this end, we have been studying the performance of several approximation algorithms, including the MST and greedy algorithms, from both a worst-case and average case point of view. A prerequisite for our evaluation of the average case performance, has been the development of a simple probability model that can yield data relevant to real networks. We have developed such a model and have begun using it to evaluate the MST, greedy algorithms and others. We have shown experimentally, that the average case performance of the MST algorithm is excellent, usually within 5% of optimum. While this algorithm is probably impractical for application in a real network, our results show that it can serve as a useful standard of comparison against which other algorithms may be measured. In particular, we have used it to study the performance of the greedy algorithm in dynamically changing connections. Our results show that the solutions produced are generally within 20% of the value obtained for the MST algorithm. The performance deteriorates during long sequences of deletions, because the algorithm simply prunes rather than re-routing during such sequences. This sort of degradation is not unique to the greedy algorithm, but is intrinsic to any algorithm that makes only incremental

changes and does not re-route.

Our research plans include continued experimental evaluation of these algorithms and others. We have also begun to study the average case performance of these algorithms analytically, in order to obtain greater insight into the factors limiting their performance. We also plan to design and implement distributed versions of these algorithms.

Congestion Control

A principal advantage of packet switched networks is their ability to dynamically allocate bandwidth to the users who need it at a particular instant. Since networks are subject to rapid statistical variations in demand, care must be taken to ensure acceptable performance under conditions of peak loading. Congestion control refers to the collection of methods used to ensure each user acceptable performance under a variety of load conditions. The high speed and multi-point connection capability of broadcast packet networks place new demands on congestion control methods.

A prerequisite to the development of an effective congestion control method is an understanding of the impact that bursty sources have on queueing in the network. The popular M/M/1 queueing model, while theoretically tractable and widely applicable, is insufficient to model the behavior of a small number of high speed and very bursty sources. A key part of our work in congestion control has therefore been to obtain an understanding of such sources. We are focussing on a simple model that treats each source as a two state Markov chain. The source is active in one state and idle in the other. Parameters of the model include average holding times in each state and the rate of packet transmission while active. This model can be used for a wide variety of bursty sources, including coded video. Our results to date indicate that such sources can lead to serious performance degradation if not handled carefully.

The basic congestion control mechanism under consideration involves user specification of several parameters defining peak and average bandwidth requirements, plus a measure of burstiness. The network uses these parameters to calculate an *effective bandwidth*, which is used for allocating link bandwidth. In the past six months we have developed a candidate method for computing effective bandwidths. This is based on the Markov chain analysis mentioned above, along with an interpolation scheme to permit rapid calculation in a realistic network context. We have found that for sources with peak and average bandwidths of more than a few percent of link bandwidth, the effective bandwidth is quite sensitive to how bursty the connection is, but for lower values, it is fairly insensitive. One implication of these studies is that to achieve effective bandwidths substantially lower than peak

for bursty, high speed sources we must either increase link speeds, buffer sizes or both.

The network also ensures that individual users don't exceed their specified rate, using a simple *traffic valve* at the edge of the network. One simple implementation of a traffic valve can be viewed as a *pseudo-buffer* for which the user specifies the peak arrival rate, the serving rate and the buffer size. Whenever the user sends a real packet, the network adds a pseudo-packet to the pseudo-buffer. If this does not cause the pseudo-buffer to overflow, the real packet is immediately accepted by the network. Otherwise it is discarded. (Note that only pseudo-packets go into the pseudo-buffer.) This mechanism is simple enough to be implemented within packet processor chips at the boundary of the network.

For multi-point connections with several transmitters, additional complications arise, since the control of entering traffic provided by the traffic valves at the edge of the network allows excess traffic on internal links of multi-point connections. We have designed a mechanism to control this kind of overload, which in effect allocates link buffer space in direct proportion to bandwidth allocations, and discards packets belonging to connections that exceed their share. This mechanism, in combination with others we have developed, allows a general solution to the problem of multi-point congestion control.

We have also considered a different approach to multi-point congestion control, in which the network actively controls the number of simultaneous transmitters in a multi-point connection, rather than limiting itself to the protection of its internal resources. This kind of access arbitration could be more attractive to users, as it regulates the flow of traffic on a channel in a more consistent fashion. We have developed two general approaches to access arbitration, and several specific algorithms.

Packet Video

Packetized transport of video signals raises a variety of important issues that we are beginning to explore. One major effect of packet transport on video coding is to eliminate the constraint of a constant bandwidth channel that currently drives most work in video coding. A variety of techniques including transform coding, motion compensation, differential coding and adaptive quantization are currently used to reduce the required bandwidth for video signals. Existing systems use buffering and variable rate coding, with the objective of achieving minimum image distortion for a given, fixed channel bandwidth. In the context of packet transport, we can exchange the objective function we seek to optimize with the constraint. That is, we code to achieve minimum bandwidth subject to a given constraint

on distortion. This approach allows the bandwidth to vary across a wide range, achieving low average bandwidths and high picture quality.

Packetized transport also raises the issue of picture quality in the presence of packet loss. Common video coding methods rely heavily on state information that can become inconsistent when data is lost. The impact of lost packets can be reduced by interpolation schemes, in which a given block of information is split across multiple packets, allowing partial recovery of lost information. We expect that the use of such methods in combination with low rate transmission of complete state information can maintain high picture quality in the face of substantial packet loss rates and we are studying such methods to assess their potential.

Historically, video coding methods have been used primarily to produce moderate quality video for conference applications. With high speed packet networks it may also be advantageous to apply video coding methods to very high resolution signals; the objective becomes not bandwidth reduction but higher resolution. This raises some interesting issues in codec design. Current codecs can be built with limited parallelism because of the low resolutions and data rates that they must cope with. Codecs for high resolution video will require greater parallelism and greater reliance on custom integrated circuits.

Administrivia

In the past year, we have grown from a small base to a research team that now includes two faculty members, one full-time staff person, one visiting research associate and nine graduate students. Additional faculty are also being recruited in both the Computer Science and Electrical Engineering departments, and this will have an important impact on the project. One recent addition to the Computer Science faculty is Gurudatta Parulkar who has just graduated from the University of Delaware. Dr. Parulkar's thesis research focussed on the design and analysis highly reliable local area networks based on flooding protocols. We expect him to be an important collaborator for the ACS project.

Our funding picture is fairly healthy. In addition to the support we receive from our three corporate sponsors, we have a major grant from the National Science Foundation that currently provides about 60% of our funding. In addition to the direct grant support, NSF provides access to MOSIS, their silicon fabrication service which we are using heavily in our prototyping effort. We have recently benefitted from a change in the policy of the Washington University School of Engineering and Applied Science; the school now pays the tuition of graduate students on research assistantships rather than requiring the research grant to pay that portion. This change has allowed us to increase our graduate student stipends

| Name | Degree (exp. graduation date) | Research Area |
|--------------------|-------------------------------|------------------------------|
| Shahid Akhtar | MS (10/87) | congestion control |
| Neil Barrett | MS (5/89) | communication circuit design |
| Victor Griswold | DSc (1/90) | connection management |
| Mark Hunter | MS (5/88) | connection management |
| Shabbir Khakoo | MS (5/88) | packet video |
| George Robbert | MS (5/88) | CAD tools |
| James Sterbenz | DSc (1/90) | communication circuit design |
| Bernard Waxman | DSc (1/89) | routing |
| Einir Valdimarssen | MS (5/89) | communication circuit design |

Figure 1.3: Graduate Student Staff

which have been low with respect to other schools with which we compete. We have also used some of the funds made available by this change to improve our base of computing equipment.

While the project's funding situation is in fairly good shape at the moment, we anticipate that additional funding will be required if we are to achieve all our major goals. The most likely source of new funding in the short term is through expansion of the Consortium to five or six members. We are currently exploring the possibility of consortium membership with three new companies. Such expansion could provide adequate funding through September 1989. After that time, substantial new funding may be required. We are beginning to explore possible sources of that funding including an NSF Engineering Research Center grant.

The project currently supports professors Jonathan S. Turner and Mark Franklin (part-time) plus eight graduate research assistants (see Figure 1.3). We have one additional student (Akira Arutaki) who is supported by NEC. We expect to add between one and three additional graduate students in the coming year. In addition, we have one professional staff member (Pierre Costa) and one visiting research associate (Riccardo Melen).

For administrative purposes, the ACS Project operates within the Computer and Communications Research Center directed by Professor Mark Franklin. The Center has a central office suite housing professors Franklin and Turner, plus eight graduate students, on the third floor of Bryan Hall, across from our main laboratory facility. This laboratory houses our main computers, and a cluster of terminals and workstations for graduate student use and also serves as an informal meeting room.

We have a second laboratory on the fifth floor of Bryan which is devoted to our hardware prototyping efforts. Over the summer, we acquired additional space on the fifth floor of Bryan adjacent to the laboratory, which has been converted to office space.

The Center's base of equipment includes a VAX 750, a MicroVax II/GPX and a Sun 3/280, all running Unix. The MicroVax is used primarily to support VLSI design work. The Sun is a recent addition and is used primarily as a file server for five Sun 3/50 workstations we have also just acquired. We have about fifteen conventional terminals, a second VLSI design station, and assorted printers. We also have assorted lab equipment including a Tektronix logic analyzer and IC tester.

We have been generally successful in expanding the Center's space and facilities to meet our needs. As we are not planning substantial additional growth in the immediate future, we feel reasonably comfortable with the current situation. On the other hand, space shortages may develop in the next year as the Computer Science and Electrical Engineering departments continue to expand their faculties. While the Engineering School is planning a new building which will relieve the space shortage in the long terms, temporary steps will undoubtedly be required in the interim.

2. Switch Architecture Studies

| | |
|--------------------|-----------------|
| Faculty | Mark Franklin |
| | Jonathan Turner |
| Research Associate | Riccardo Melen |
| Graduate Student | James Sterbenz |

The architecture of high speed packet switching fabrics is of course central to the work of this project. While we are concentrating our efforts on a particular design [59], we continue to evaluate alternatives, in order to identify possible improvements. In the past year, we have made a close study of two alternative architectures for fast packet switching that are being developed by other research groups. Our objective in this study has been to understand the similarities and differences among the different fabrics and make a preliminary assessment of relative strengths and weaknesses. We hope to use the insights gained in this way to obtain better designs.

This chapter has three main sections. The first summarizes our comparisons of the broadcast packet switching fabric [59] with the Starlite [26,27,28] and Prelude [7,14,22] switch fabrics. The second outlines some possible improvements to the broadcast packet switch fabric that were suggested by our comparative studies. The third outlines some of the issues associated with constructing very large switching systems from modules of moderate size. We close with a summary and brief discussion of our research plans.

2.1. Comparison of Alternative Switch Fabrics

The switch fabric for the Broadcast Packet Network (BPN), described in [59] is based on buffered binary routing networks. It is topologically simple and well-suited to VLSI implementation. The overall structure is shown in Figure 2.1. The system consists of a set of *Packet Processors* which interface to the external links and provide all per packet protocol processing, a *Connection Processor* which sets

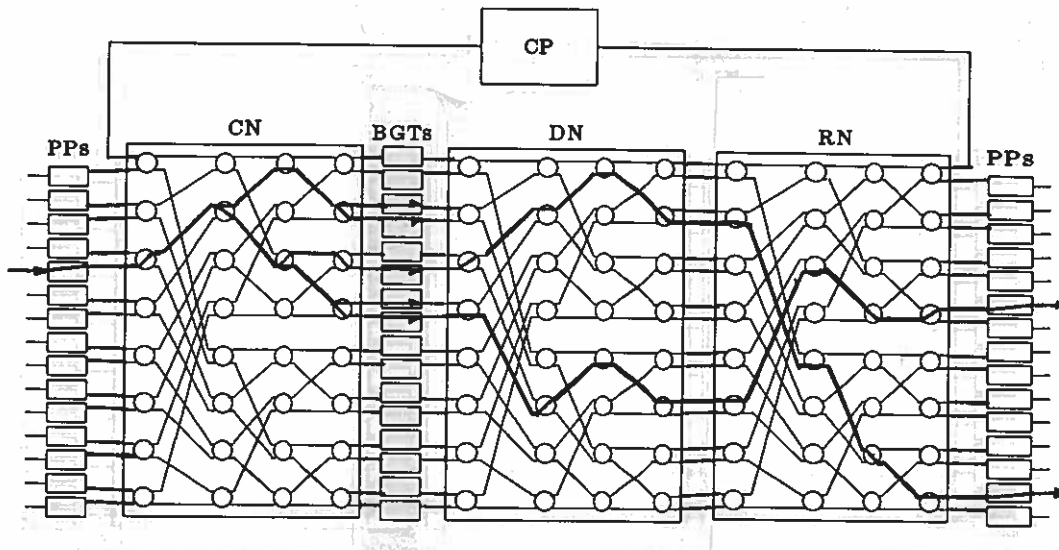


Figure 2.1: Broadcast Packet Switch Fabric

up and maintains multipoint connections, and a switch fabric consisting of a *Copy Network*, a set of *Broadcast and Group Translators*, a *Distribution Network* and a *Routing Network*.

Packets enter one of the Packet Processors at left, where an address translation is performed. For point-to-point packets this yields an outgoing link number and an outgoing channel number. These are placed in the header of the packet, which then passes through the CN, one of the BGTs and the DN, following some arbitrary path. When the packet reaches the RN, it is routed using the outgoing link number. The RN is a conventional binary routing network with sufficient storage at each node to store a small number of complete packets. When the packet reaches the outgoing PP, the extra header information added at the incoming PP is stripped off and the packet is transmitted on the outgoing link. The role of the DN is to randomly distribute packets it receives across its outputs. This prevents congestion that can otherwise occur in the RN when subjected to traffic patterns with strong "communities-of-interest."

When a packet belonging to a multipoint connection is received at an incoming PP, it undergoes a similar translation process, but the information added to the packet header is different. It consists of two fields, a *Fanout* field which specifies the number of outgoing links which are to receive copies of the packet, and a *Broadcast Channel Number*, used by the BGTs. The CN replicates multipoint packets using the fanout field to guide its decisions. At each switch element where replication is performed, the fanout fields of the two copies are modified (essentially by halving the original fanout), so that a short time after the original packet enters the CN,

the appropriate number of copies appears at its outputs. The BGTs then perform a translation similar to that done in the PPs, using the broadcast channel number in the copies to index a table, yielding a set of outgoing link and logical channel numbers. These are added to the packet header and used to guide the copies to the proper outgoing links.

This design is well-suited to implementation in a medium speed, high density technology like CMOS. While the per node buffering makes the individual switch elements moderately complex, the topological complexity is very low. The only large memories are in the PPs and BGTs, and these need be accessed only once per packet cycle, permitting the use of high density memories with relatively long cycle times.

In the last five or six years, several experimental switching system designs have been proposed that can support multirate and multipoint communication in a flexible fashion. In addition to the BPN fabric being designed at Washington University, there is the Starlite system originally developed by Alan Huang and Scott Knauer at AT&T Bell Labs and currently being developed further by a group at Bell Communications Research, and the switching matrix for the Prelude experimental wide band switching system, developed by Coudreuse et. al. at CNET in France. Starlite and Prelude are described below.

Starlite

Starlite is the name given to an experimental switching system developed by Alan Huang and Scott Knauer at AT&T Bell Labs [26,27,28]. The Starlite architecture was motivated by the observation that sorting networks, can be used to construct rearrangeably non-blocking switching fabrics with distributed control. This observation was first put forward by Batcher [2] in 1968 in his seminal paper describing his *bitonic sorter* that sorts a set of n numbers using a network of approximately $(n/4)(\log n)^2$ simple comparison elements. For circuit switching applications, this observation leads to switching networks that are non-blocking, operationally very simple and eminently suited to VLSI implementation. To accommodate packet switching, mechanisms are needed to resolve contention between packets that arrive concurrently and are destined for the same output port. Multipoint communication requires additional mechanisms for packet replication. Huang and Knauer's contribution was the development of inexpensive VLSI implementations of Batcher's sorting network and the invention of a variety of supplementary networks which support packet switching and multipoint communication when used in concert with the sorting network. More recently, a group at Bell Communications Research has adopted Starlite as the basis of a major applied research effort in high speed packet switching and have devised a number of improvements. In this section, we first

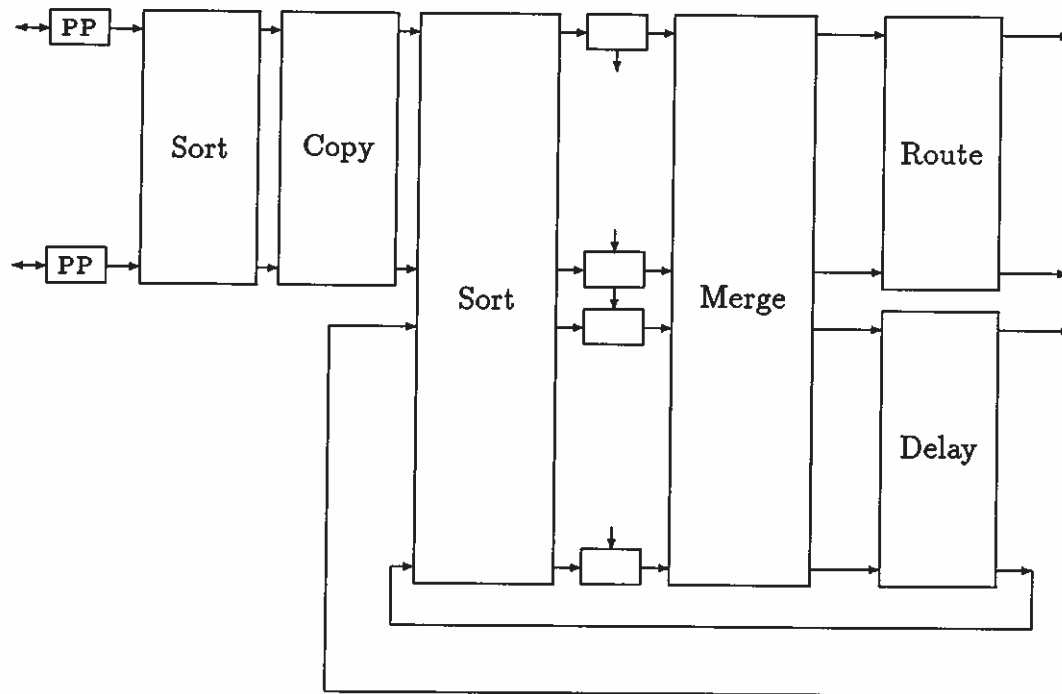


Figure 2.2: Starlite Switch Fabric

describe some of the original work on Starlite done at Bell Labs. We will then review some of the more recent improvements.

While Huang and Knauer made no serious attempt to develop complete systems, they did develop a variety of useful tools that can be used for the construction of such systems and suggested ways in which they could be used. Figure 2.2 shows one possible implementation of a packet switch supporting multipoint communication. Packets arriving on external links enter a set of *Packet Processors* at left, which perform some address translation. For point-to-point packets this results in a destination PP number being placed in the packet header. This is used to guide the packet to the appropriate outgoing link.

For the moment, we will ignore the initial sort and copy networks at the top left and concentrate on what happens to packets when they enter the main sorting network at the middle of the figure. This network sorts packets in increasing order of their destination addresses, meaning that when the packets exit the sorting network, all packets with same destination address occupy a contiguous set of output links. The filters at the exit of the sorting network mark all but one packet destined for a particular address, by comparing the destination addresses

of packets on adjacent links; if a packet has the same destination address as the packet on the next lower link, its *wait bit* is set. Packets for which the wait bit is zero are forwarded to the routing network at right which routes them to the proper outgoing links. Packets with the wait bit set are sent to one of a set of delay elements, which delays them for approximately one packet time, after which they are recirculated through the sorting network. It's useful to extend this basic scheme by adding a second field to each packet which records the number of times a packet has recirculated. By having the main sorting network use this field as a secondary sort key, we can also order packets by their age, giving older packets priority over newer ones. This ensures that packets are transmitted in the same order in which they were received. If the network supports n external links and the main sort and merge networks have m input and output ports, up to $m - n$ packets may be recirculating at any time. Packets may be lost if during a cycle, more than $m - n$ packets have their delay bits set. The value of m is selected based on statistical considerations, to yield an acceptably low probability of packet loss.

We now turn to the issue of packet replication. The network is designed around the notion of a coordinated copy between source and destinations. That is, the source and the destinations must synchronize when a packet is to be copied. When the source PP sends a packet into the network, the destination PPs simultaneously send *blank packets* containing their address plus the address of the source in the headers. The initial sorting network sorts these packets on the source addresses, which places the original packet and associated blank packets on a contiguous set of links, upon exit from the sorting network. The copy network then copies the information from the source packets to each of the blank packets, a relatively straightforward process, given the sorted arrangement. When these packets enter the main sorting network, they are routed using destination addresses in the same way as point-to-point packets.

The Starlite system has some very attractive properties. The basic switch elements making up the various networks are simple and have a regular interconnection pattern, which makes the design of high speed VLSI implementations quite straightforward. The network is non-blocking and has a latency of only one bit time per stage of switching. It maintains packet sequencing, so that packets are received in the same order in which they are transmitted. The sharing of buffering across the switch fabric, rather than dedicated it to individual links provides more predictable performance in the face of statistical fluctuations in traffic.

The synchronization required for copying is a drawback of this approach, when used in a general packet switching environment. Some form of arbitration is required at the front end to schedule packets that must be copied, and while such a mechanism is probably feasible, no detailed proposal has been put forward. Also, while the switch elements are very simple, their interconnection is topologically

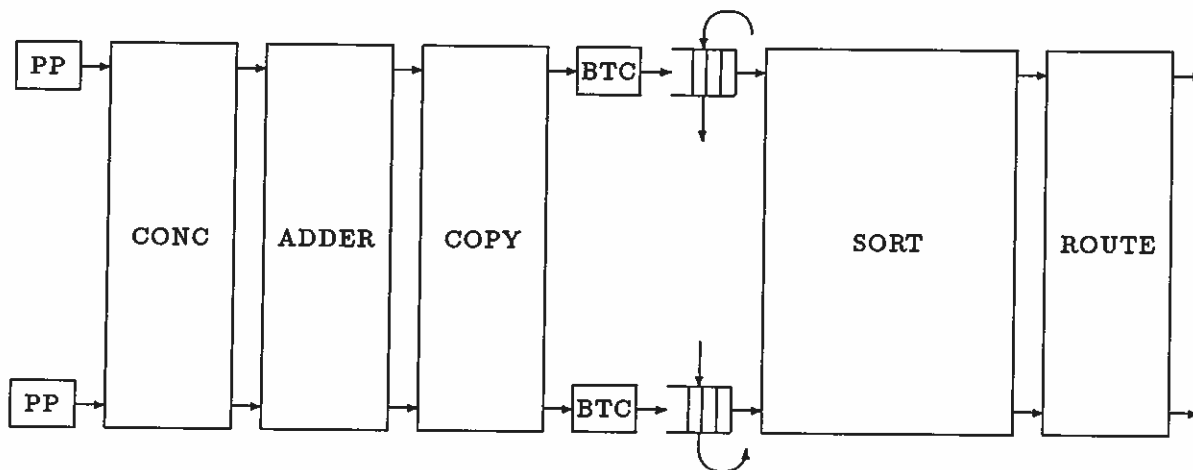


Figure 2.3: Alternative Starlite-Type Switch Fabric

complex relative to competing proposals. Finally, the dimensioning of the main sort and merge networks is problematical; it appears likely that to achieve satisfactory performance in a general packet switching environment, these networks must have at least four to eight times as many inputs as there are external links. Nevertheless, the Starlite approach is a very promising one, and is a convincing demonstration of the power of a few simple ideas.

The many attractive features of Starlite have led a group of researchers at Bell Communications Research to use it as the basis for their work on high speed packet switching. This group has substantially extended the work of Huang and Knauer in a number of ways and have developed better or more complete solutions to the problems of output contention and packet replication. We briefly describe one possible system configuration based on their work. The reader is referred to references [9,38] for further details.

Figure 2.3 shows a switching fabric with several components. Packets are received at the Packet Processors (PP) on the left where they are assigned a fanout and broadcast channel number, as in the BPN fabric. The packets then pass through a concentrator network (consisting of $n \log_2 n$ simple switch elements), which places the packets on consecutive outputs so as to ensure non-blocking operation of subsequent networks. Next, the packets pass through a running adder network, which for the packet on output port i computes the sum of the fanouts of all packets entering on ports 0 through $i - 1$ and places this sum in a field of the packet (this is done for all output ports, using a network with $n \log_2 n$ simple processing elements). Following the adder, the packets pass through a copy network which uses the fanout and the sums generated by the concentrator to generate the proper number of copies. Note that during any one cycle, the packets entering may have

a total fanout larger than n . The system accommodates as many packets as are feasible and the PPs resubmit 'losing packets' on the next cycle. Note that there is no internal buffering in the fabric. The specific mechanisms employed ensure that blocking cannot occur due to contention for internal paths within the fabric.

After the packets emerge from the copy network, they are processed by a set of broadcast translation circuits, which assign outgoing link numbers to each packet, as in the BPN system. Unlike the BPN system, each broadcast translation circuit here must be able to route any one of the copies of a broadcast packet, implying a substantial expansion in the memory requirements.

The packets then enter a set of queues, followed by a sorting and routing network. The sorting network routes packets on destination address and the routing network then routes them to the proper output once they are in sorted order. The queues are interconnected by a *control ring* which arbitrates access to switch fabric output ports. The control ring has a bit for each output port and a queue which has a packet to send to a particular output port signals its intention by setting the bit corresponding to that port. Since the control ring circulates through the queues sequentially, only one queue is permitted to send a packet to a particular output port during a cycle. The use of the control ring eliminates output contention and consequently, the potential for internal congestion in the routing network.

This configuration offers a complete solution to the problem of multipoint packet switching. The most complex single element is the sorting network; the other components together are roughly equal to the sorting network in complexity for networks with 1024 input and output ports. While substantially less complex than the first architecture described, it lacks the advantage of shared buffering. It also introduces the potential for packets to be misordered, since consecutive packets in a connection may be placed in different queues following the BTCs.

Prelude

The Prelude project began at CNET in France in the early eighties, with the objective of creating a flexible switching system that could provide point-to-point and multipoint communication at speeds up to a few hundred megabits per second [7,14,22]. It is based on a particularly simple form of fast packet switching referred to as *asynchronous time division multiplexing*, and uses a novel high speed switch fabric.

The basic structure of the Prelude switch fabric is shown in Figure 2.4. Packets enter at the transmission interfaces at left, which perform framing and synchronization functions. The packets are then passed through a rotative switch which transforms each packet to a so-called parallel-diagonal format (*paragonal*) in which

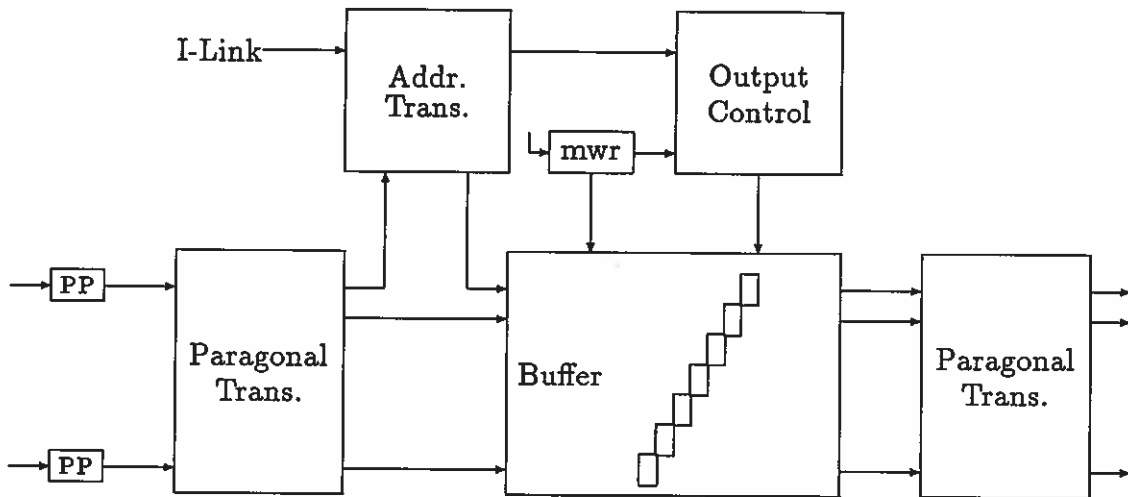


Figure 2.4: Prelude Switch Fabric

each packet is distributed across the outputs of the rotative switch, with the first byte of each packet on the first output, the second byte on the second output, and so forth. This transformation places the headers of all packets on the first output of the switch, where they can all be processed by a centralized address translator. The address translator modifies the channel number in the header of the packet and then the modified packet is stored in a central buffer memory, still in the paragonal format. At the same time, the address translator passes to the output control circuit, a bit vector defining which outputs are to receive copies of the packet. The output circuit stores the address at which the packet header was written, in queues associated with the selected outputs. This information is used later to retrieve the packet from the central buffer. There is an output process that examines these queues in a cyclic fashion, initiating a new packet retrieval on each clock cycle. Broadcast is accomplished simply by reading the packet from the buffer once for each output that requires a copy. Note that these reads need not all take place during one packet cycle. Finally, a second rotative switch transforms the packets from the paragonal format back to the normal format so that they can be output on their respective links.

This design has several attractive features. The basic elements are simple; the rotative switches can be implemented as barrel shifters, requiring about $n \log n$ gates, the address translator and buffer are essentially just random access memories with a modest amount of control circuitry, and the output control consists of a fairly simple and regular collection of queues and address registers. As with Starlite it maintains packet sequencing and provides a single shared buffer rather than

per line buffers. It is, on the whole simpler than the Starlite fabric and handles multipoint communication in a more satisfactory way.

The main drawback of this approach is its dependence on high speed memories, particularly in the central buffer. It must be possible to access this memory twice per clock time, once for reading and once for writing. There does not appear to be any architectural way to reduce the required memory cycle time for individual memory chips since the memory read-out process can access the memory chips in random order. Another drawback is that since channel translation takes place before packets are replicated, all the downstream copies of a multipoint packet carry the same channel number. This places operational restrictions on the assignment of channel numbers, that may be problematical, depending on the number of channels and multipoint connections. It is most troubling for general multipoint connections in which there are several transmitters. It appears that either all the links involved in such a connection must use the same channel number, or there must be a different channel number for every incoming port that can be the source of the packet. The latter solution requires that the downstream switches treat all those channel numbers similarly.

Remarks

One major difference between the BPN approach and Starlite and Prelude, is the use of buffering within the switch fabric. While this leads to more complex switch elements, the added complexity has little impact on cost or performance. The reason is simply that cost is determined primarily by component count, which because of pin limitations is determined primarily by topological complexity, not circuit complexity. While simple switch elements are better suited to low density technologies such as ECL, they have little advantage in the context of a high density technology like CMOS. Furthermore, the packet processors required in all three systems, have inherently high circuit complexities and must be implemented in high density technologies to be economical.

Starlite and Prelude do have two potentially significant advantages over the BPN approach. First, they can both implement shared buffering, leading to more predictable performance in the presence of highly bursty traffic. While the addition of some shared buffering to the BPN fabric is feasible, no detailed study of such an arrangement has yet been made.

Starlite and Prelude also guarantee that packets are transmitted in the same order as they are received. While it is possible to modify the BPN design to provide a similar guarantee, the required changes may impose operational constraints and degrade performance.

This brief review leads to three conclusions. First all three of the switch fabrics reviewed are viable architectures; they can all be used to support high speed packet networks and multipoint communication in an effective way. Second, none clearly dominates the others; each has a different set of advantages and drawbacks. Third and perhaps most important, each offers some useful lessons to architects of future systems. In the next section, we consider some of the ways in which these lessons might be applied.

2.2. Refinements to the BPN Switch Fabric

We now briefly consider several key architectural issues that will be addressed in the coming year. The first is the problem of resequencing packets; second is the issue of shared buffering; third, a potential application of partial sorting to the BPN fabric; and fourth, some issues that arise in the design of very large systems.

Resequencing

The design of the BPN switch fabric allows packets belonging to a particular logical channel to exit the fabric in a different order from the order in which they are received. We briefly consider several alternative methods for dealing with this problem.

The first method is not to allow packets to get out of order in the first place. This is the approach taken by Starlite and Prelude and can be adopted with the BPN fabric. The key is to distribute traffic in the CN and DN on a per channel basis rather than a per packet basis. This complicates the hardware slightly and the control software substantially. It also introduces the possibility of a connection blocking when a path with sufficient capacity cannot be found. This approach is being taken by a group at CSELT in their fast packet switching research project.

A second method is to allow resequencing to be done on an end-to-end and application-dependent basis. Preliminary performance data indicate that the probability of packets arriving out of order is extremely unlikely for applications with peak data rates of less than 20% of the FOL data rates. The reason is simply that the time between arrivals of successive packets at a switch is larger than the time it takes a packet to propagate through the switch fabric. Only applications with very high peak data rates are expected to experience a significant rate of packet misordering. This implies that the majority of applications can ignore the resequencing problem and suggests that the provision of a general mechanism for handling it may be unwarranted. Applications such as video or file transfer can

handle the resequencing problem in a simple and straightforward way (for video, for example one merely places received packets in the proper position in the frame buffer based on a sequence number).

Resequencing can also be done on an end-to-end, application-independent basis. The most general solution requires buffering and a retransmission protocol. A simpler method is possible if one is willing to tolerate a small but non-zero probability of misordered packets (e.g 10^{-6}). To provide such a guarantee, one adds sequence numbers to packets as they enter the network and uses the sequence numbers to resequence packets on exit. The resequencing device buffers packets as necessary, but never holds a packet longer than a specified time. The amount of buffering required and the timeout value depend on the misordering probability of the network and on the target residual misordering probability. Resequencing can also be done on a per switch basis. The method is similar to the end-to-end, application-independent method but is somewhat simpler to implement.

Detailed evaluation of these options has not yet been attempted. The circuit complexity and performance of end-to-end and per-switch resequencing in particular, requires closer study.

Shared Buffering

While the FOL switch fabric contains buffers, which can in some sense be viewed as shared, most of the buffering actually occurs in the outgoing packet processors. For traffic with Poisson arrival statistics, per-link buffers with 64 buffer slots are sufficient to achieve packet loss rates (due to buffer overflow) of well under 10^{-6} at link occupancies of 80%. Unfortunately, many real applications have arrival statistics that are highly non-Poisson. For these bursty applications, larger buffers or lower link occupancies are required for satisfactory packet loss rates.

One way to achieve the effect of larger buffers is to provide the buffering on a shared basis as done in Prelude and Starlite. For the BPN fabric, a set of shared buffers can be placed between the distribution and routing networks, with a hardware mechanism used to control the flow of packets from the shared buffer. A simple way of implementing this control is to provide a simple TDM control ring, that carries one bit of flow control information from each PP to each of the shared buffers. This is similar to the use of the control ring in the alternative version of the Starlite fabric described above.

While shared buffering appears to be worthwhile, the situation is really less clear than it might seem. One of the advantages of per-link buffering is that it puts an upper bound on the queueing delay at each switch. In a high speed packet switching system, it appears desirable to have a maximum per-switch delay in the

neighborhood of a few milliseconds. This means that the amount of buffering per link is a few hundred kilobits (assuming 100 Mb/s link speeds), not an unreasonable amount to provide on a per-link basis. A detailed evaluation of the cost and advantages of shared buffering remains to be done.

Partial Sorting

The Starlite system suggests a possible improvement to the BPN fabric design. The routing network used in the Starlite design is a form of binary routing network (specifically, a banyan network) that will pass a sorted sequence of packets without conflict. It will also pass without conflict, a set of input packets that arrive on an input port with the same port number as their destination.

Suppose we replace the distribution network in the BPN fabric with a network that routes packets whenever it can do so without delaying them, and distributes packets otherwise. Intuitively, such a network appears likely to present the routing network with a better traffic pattern than is obtained by simply randomizing the traffic; this could lead to substantially higher throughputs. We plan to examine such a strategy in detail in the next few months and quantify the resulting performance advantage.

2.3. Design of Large Systems

The BPN switch module has been designed as a component that can be used to construct the large switching systems required for supporting ubiquitous public networks. We have formulated a possible design for a local switching system, in order to obtain a better idea of the scale of the system and the associated control issues.

The proposed design supports up to 65,000 access lines and 4096 trunks, and is based on a hypothetical traffic mix¹ including 200 entertainment video sources, video telephones with an effective bandwidth of 8 Mb/s and busy 20% of the time, plus other traffic that is equivalent to the video telephone traffic in total resource requirements. For the broadcast video, it is assumed that a 90:10 rule applies; that is, in any group of users, 90% will be accessing 10% of the available channels. This, along with the assumptions on video phone traffic were used to select concentration ratios.

¹It should be noted, that this is purely hypothetical and not supported by any detailed traffic projections. The purpose is just to present a rough picture of a possible traffic mix, and how one might configure a system to support it.

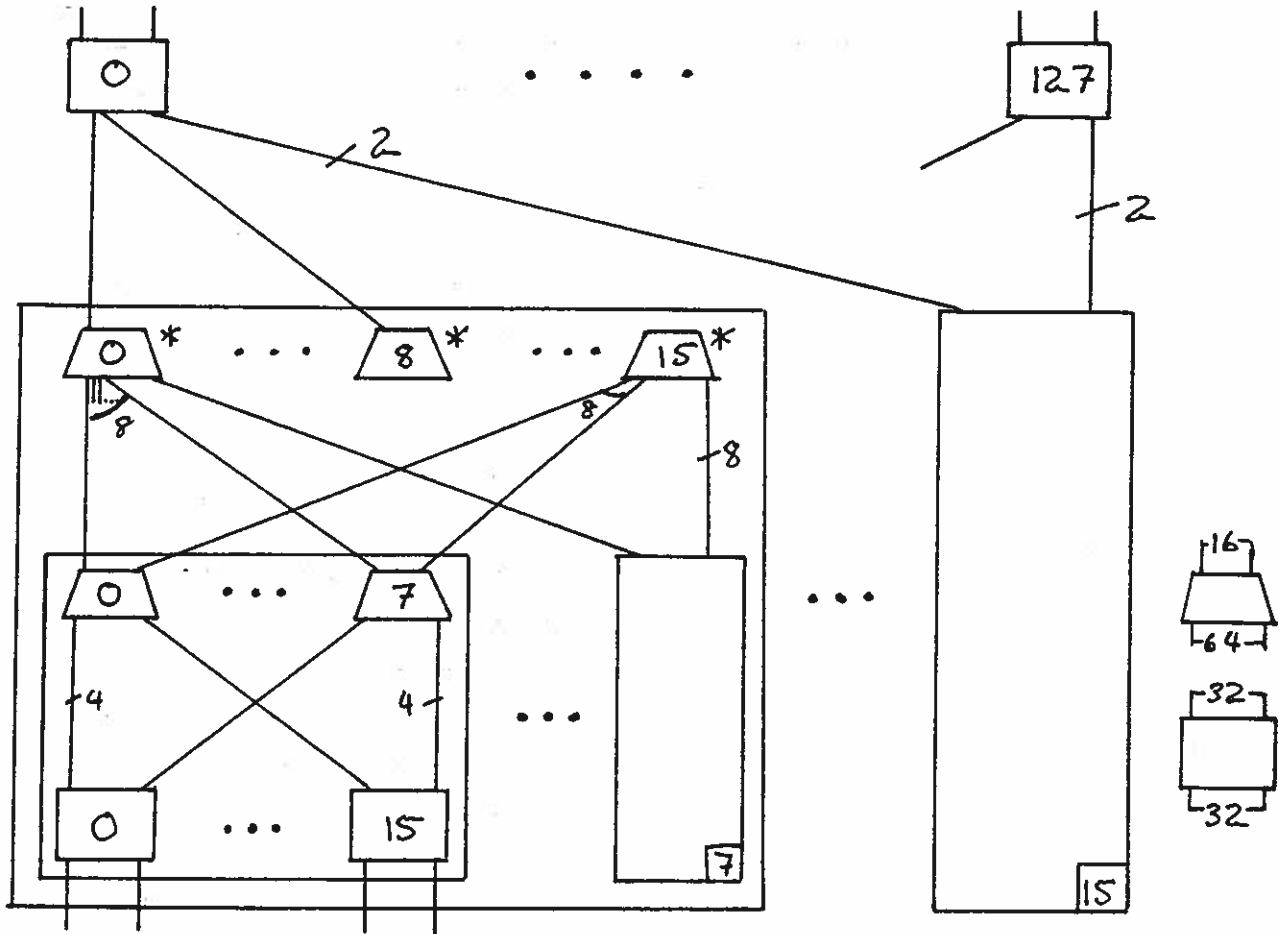


Figure 2.5: Local Switch Design

A block diagram of the system appears in Figure 2.5. The lines terminate at the bottom of the figure, trunks connect at the top. It is constructed from two types of switch modules, one with 64 links and the other with 80, and providing 4:1 concentration. The system has 128 *back-end switch modules* that connect to 4096 trunks and provide access to 16 *major groups*, which terminate 4096 lines each. The major groups each contain 16 back-end switch modules and 8 *minor groups*, terminating 512 lines each. The minor groups each contain 8 back-end switch modules and 16 front-end switch modules.

The hardware complexity of this system can be estimated based on the assumption that one unduplicated switch module can be implemented on 8-12 printed circuit boards. If the front-end switch modules in the minor groups are unduplicated and everything else is duplicated, a minor group requires two equipment frames, a major group requires 18 frames and the complete switch requires 304 frames. This

is roughly comparable in complexity to a telephone central office supporting the same number of lines.

This design raises several questions. First, one might ask, why structure the system in this way in the first place? Why not instead, simply scale the switch module to a much larger size? This turns out to be impractical for a variety of reasons, including expandability, timing, reliability and control partitioning. A large system of this sort must be broken down into a large number of fairly small, independent modules. This raises an interesting point with respect to the Starlite and Prelude switch fabrics. Those fabrics, like the BPN fabric, are useful mainly for constructing switch modules of moderate size; to build very large systems they must be configured together into a larger structure. In that context, many of the apparent distinctions among the different approaches become less significant.

Another question that arises is how connection management is handled in a large system. The simplest method conceptually is to allow the connection processors in each switch module to operate independently. While this can be workable, it requires more effort and more delay in the connection establishment process than is strictly necessary. It is possible to speed up the process considerably by using datagram routing within the switch, limiting the logical channel-based routing to the external lines and trunks. The regularity of the interconnection topology makes the implementation of datagram routing straight-forward for point-to-point connections. It remains unclear whether such an approach is practical for multipoint connections as well.

Using datagram routing on the inter-module links eliminates the need for most internal resource allocation, but still leaves the problem of resource allocation on the external links. To manage this, a database is required that tracks the status of the external links. We plan to explore the issue of how such a database should be organized to provide rapid access by the connection processors, that must use the information in the database to make routing decisions.

Summary

Studies of switching architectures have and will continue to play an important role in our work on high speed packet switching. In the last year we have made some comparative studies of switch architectures and attempted to apply the lessons learned from these studies to improve the broadcast packet switch design. In the coming year, we expect to broaden this kind of study, to consider a wider range of alternative architectures. We also want to develop quantitative comparisons, as opposed to the qualitative comparisons made here.

3. Performance Studies

Faculty
Research Associate

Jonathan Turner
Riccardo Melen

The evaluation of any switching system architecture is determined in large part by performance issues. We have addressed performance issues in several different ways. Previous progress reports have reported on extensive simulation studies examining several aspects of the performance of the proposed Broadcast Packet Switch fabric. In this report, we examine two broader performance issues. First, we consider a general method of evaluating the loading characteristics of packet switching fabrics that dynamically distribute their load across all available paths. As we shall see, such fabrics can be made robust in the face of arbitrary traffic patterns with minimum complexity. We refer to the analysis method as *fluid flow loading analysis*, and using it, we derive several fundamental results for both point-to-point and multipoint packet switching fabrics. In section 3.2, we consider a class of fabrics in which all packets belonging to a particular connection are constrained to follow the same path. The prime motivation for making such a constraint is to eliminate the possibility of packet mis-ordering. As we shall see, this consideration leads to a natural generalization of the classical theory of non-blocking networks; in this report we define that generalization, outline the important problems and present a few fundamental results. We close this chapter with a brief description of simulation results which quantify the potential for packet misordering in the proposed broadcast packet switch design. These results are intended to be used to help design a mechanism to resequence misordered packets on a switch module basis.

3.1. Fluid Flow Loading Analysis

In this section we introduce a systematic method of analyzing the effects of a given traffic configuration on packet switching fabrics that dynamically distribute

load across all available paths, and apply it to the analysis of several proposed architectures. Our method allows us to prove theorems characterizing the worst-case loading for various switching fabrics. The section gives several such theorems, both as illustrations of our method and for their inherent interest. Proofs are omitted for brevity; readers are referred to ?? for further details.

We note that the method is fairly easy to apply and leads to useful insights that can guide the switching system architect to better designs. It is not a complete characterization, as it ignores queueing and contention, but when used in conjunction with queueing and simulation models based on uniform random traffic, it can provide the designer and performance analyst with a more complete understanding of system performance.

Networks for Point-to-Point Communication

We define a *packet switching network* (or simply network) as a directed graph $G = (N, L)$ consisting of a set of nodes N and a set of directed arcs or *links* L . In addition, G contains a set of distinguished *input nodes* I and a set of distinguished *output nodes* O . Input and output nodes are also referred to as *ports*. Each input port has a single outgoing link and no incoming links, while each output port contains a single incoming link and no outgoing links.

We limit ourselves to networks in which the number of input nodes equals the number of output nodes. When we refer to an n port network, we mean a network with n input nodes and n output nodes, numbered from 0 to $n - 1$. We also limit ourselves to networks, which can be divided into a sequence of stages. We say that input ports are in stage 0 and for $i > 0$, a node v is in stage i if for all links (u, v) , u is in stage $i - 1$. A link (u, v) is said to be in stage i if u is in stage i . In the networks we consider, all output ports are in a separate stage by themselves. When we refer to a k stage network, we mean that there are k stages containing internal nodes; that is, we neglect the input and output stages.

When describing particular networks, we will find it convenient to use a composition operation. We denote a composition of two networks X_1 and X_2 by $X_1 \textcircled{h} X_2$, where h is a positive integer. The composition operation yields a new network consisting of one or more copies of X_1 connected to one or more copies of X_2 , with h links joining each pair of subnetworks. More precisely, if X_1 is an n_1 port network and X_2 is an n_2 port network then $X_1 \textcircled{h} X_2$ is formed by taking n_2/h copies of X_1 numbered from 0 to $(n_2/h) - 1$ followed by n_1/h copies of X_2 , numbered from 0 to $(n_1/h) - 1$. Then, for $1 \leq i \leq n_1$, $1 \leq j \leq n_2$, we join $X_1(i)$ to $X_2(j)$ using h links; these links connect output port $(n_1/h)m + j$ of $X_1(i)$ to input port $(n_2/h)m + i$ of $X_2(j)$, where $0 \leq m < h$. Finally, we eliminate the former input and output nodes

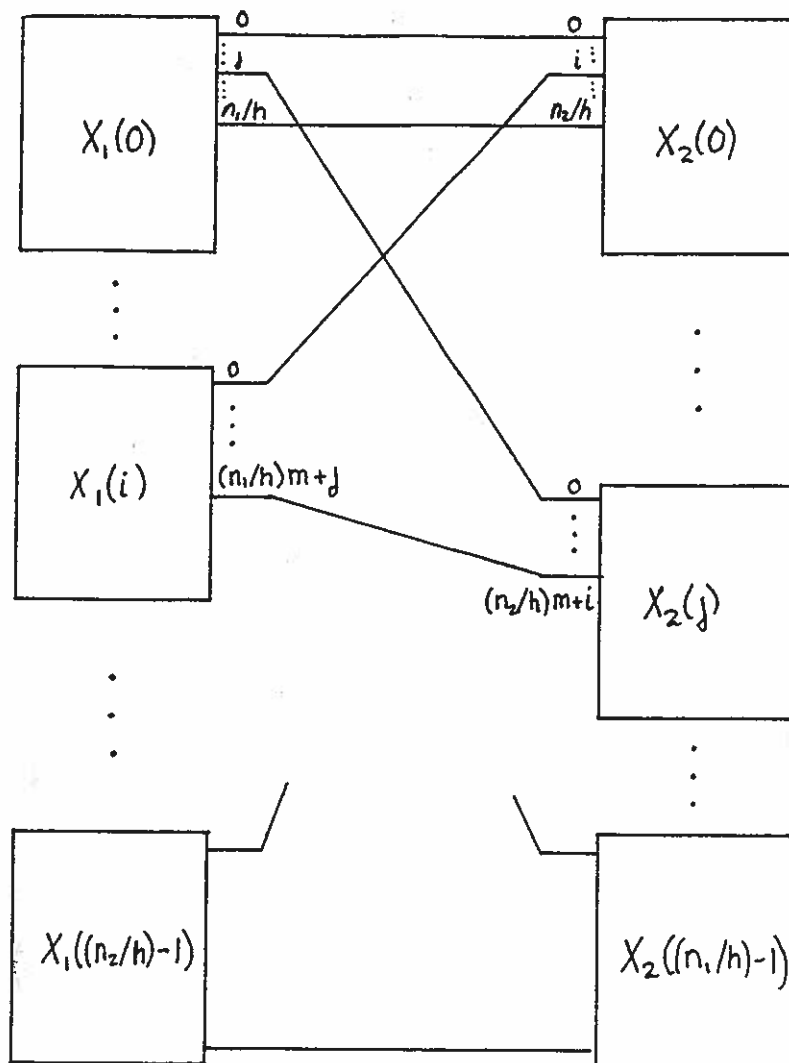


Figure 3.1: Composition Operation

that are now internal and renumber the input and output nodes of the network as follows; if u was input port i of $X_1(j)$, it becomes input $jn_1 + i$ in the new network; similarly if v was output port i of $X_2(j)$, it becomes output $jn_2 + i$. We also allow composition of more than two networks; the composition $X_1 \circledR X_2 \circledR X_3$ is obtained by letting $Y_1 = X_1 \circledR X_2$ and $Y_2 = X_2 \circledR X_3$, then identifying the copies of X_2 in Y_1 and Y_2 . This requires of course that the number of copies of X_2 generated by the two initial compositions be the same. Note this is not the same as $(X_1 \circledR X_2) \circledR X_3$. The composition operation is illustrated in Figure 3.1.

A *connection* through a network is defined as a triple (x, y, ρ) where $x \in I$, $y \in O$ and $0 < \rho < 1$. A connection *induces a load* on the various links that

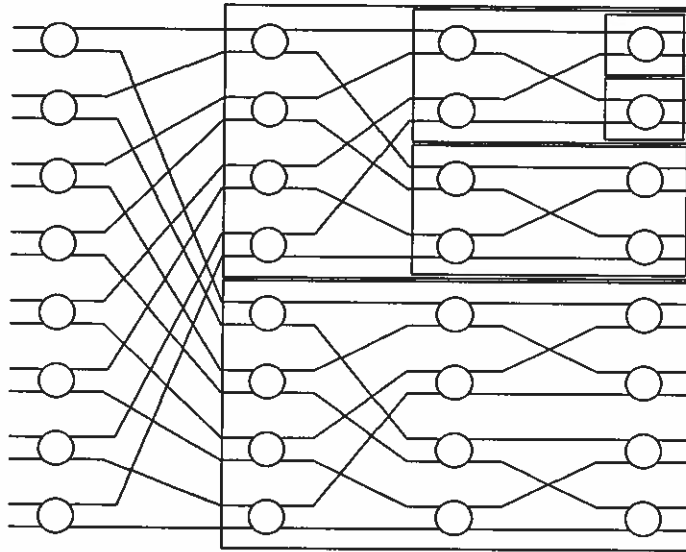


Figure 3.2: Recursive Construction of Delta Network

lie on paths joining the connection's input and output ports. The load induced by a connection (x, y, ρ) on the link leaving x is defined to be ρ . The magnitude of the induced loads on the internal links depends on the types of the nodes and the topology of the network. In this section, we will consider only a single node type. If α is the sum of the loads induced by a connection (x, y, ρ) on the input links of a node u , and u has i output links that lie on paths from x to y , then the load induced by the connection on each of these output links is α/i and the load induced on all other output links is 0.

A *configuration* is defined as a set of connections. The load induced by a configuration on a link ℓ is simply the sum of the loads induced by the individual connections and is denoted $\lambda_\ell(C)$. A configuration $C = \{c_1 \dots c_r\}$ is α -bounded if for all input and output links ℓ , $\lambda_\ell(C) \leq \alpha$. We say that a configuration is *legal* if it is 1-bounded and that a network is *robust* if for every legal configuration C , $\lambda_\ell(C) \leq 1$ for all links ℓ .

Delta networks form a well-known class of useful switching networks [11,12,13, 17]. We can define these recursively using the composition operation. Let D_1 be a network with two input ports and two output ports connected to a single internal node. We then define $D_i = D_1 \textcircled{1} D_{i-1}$ for all $i \geq 2$. We refer to D_k as a k stage delta network; note that D_k has $n = 2^k$ ports. An example of a 4 stage delta network is given in Figure 3.2.

Delta networks have been widely studied and have many interesting properties. Most useful is the *self-routing property* that allows paths from inputs to outputs to be easily determined. A related property is that there is a single path connecting any input node to any output node. For the purposes of our loading analysis, this means that a connection (x, y, ρ) induces a load of ρ on all links that lie on that path and a load of 0 on all other links. To illustrate our method of loading analysis, we start with a simple theorem which characterizes the worst-case loading for a delta network.

THEOREM 3.1.1. *Let $C = \{c_1, \dots, c_r\}$ be an α -bounded configuration for D_k . Then $\lambda_\ell(C) \leq \alpha\sqrt{n}$ for all links ℓ .*

The bound in Theorem 3.1.1 can be achieved; that is, there exist worst-case patterns that induce a load approaching \sqrt{n} on some of the internal links. We note that delta networks are readily generalized to networks in which each internal node has m input ports and m output ports. The worst-case loading in such networks is the same as for networks constructed from two port nodes.

The bound in Theorem 3.1.1 and the fact that there are traffic patterns that achieve the bound, lead to the conclusion that the binary (and m -ary) delta networks can perform poorly in the worst-case. This has been observed previously and various approaches have been proposed to remedy the situation. We review two such approaches here. The first is to add one or more stages of distribution nodes at the front of a delta network.

We denote a delta network with k routing stages and d distribution stages as $D_{k,d}$, which we define by $D_{k,d} = D_d \textcircled{1} D_{k-d} \textcircled{1} D_d$. This is illustrated in Figure 3.3. If we consider the load induced by a connection (x, y, ρ) on the links in such a network, we note that for any node u in the first d stages that lies on a path from x to y , both of u 's output links lie on paths from x to y , hence the incoming load from the connection is distributed across u 's output links. In contrast, any node v in the last k stages is on at most one path from x to y . We refer to the nodes in the first d stages as distribution nodes and the nodes in the last k stages as routing nodes.

THEOREM 3.1.2. *Let $C = \{c_1, \dots, c_r\}$ be an α -bounded configuration for $D_{k,d}$. Then $\lambda_\ell(C) \leq \alpha n 2^{-\lfloor (k+d)/2 \rfloor}$ for all links ℓ .*

The bound in Theorem 3.1.2 is the best possible; that is, there exist traffic patterns approaching the given bound. Theorem 3.1.2 tells us that every time we add two distribution stages, we reduce the worst-case load by a factor of 2. To achieve a robust network, we require $d = k - 1$. Also note that with respect to worst-case

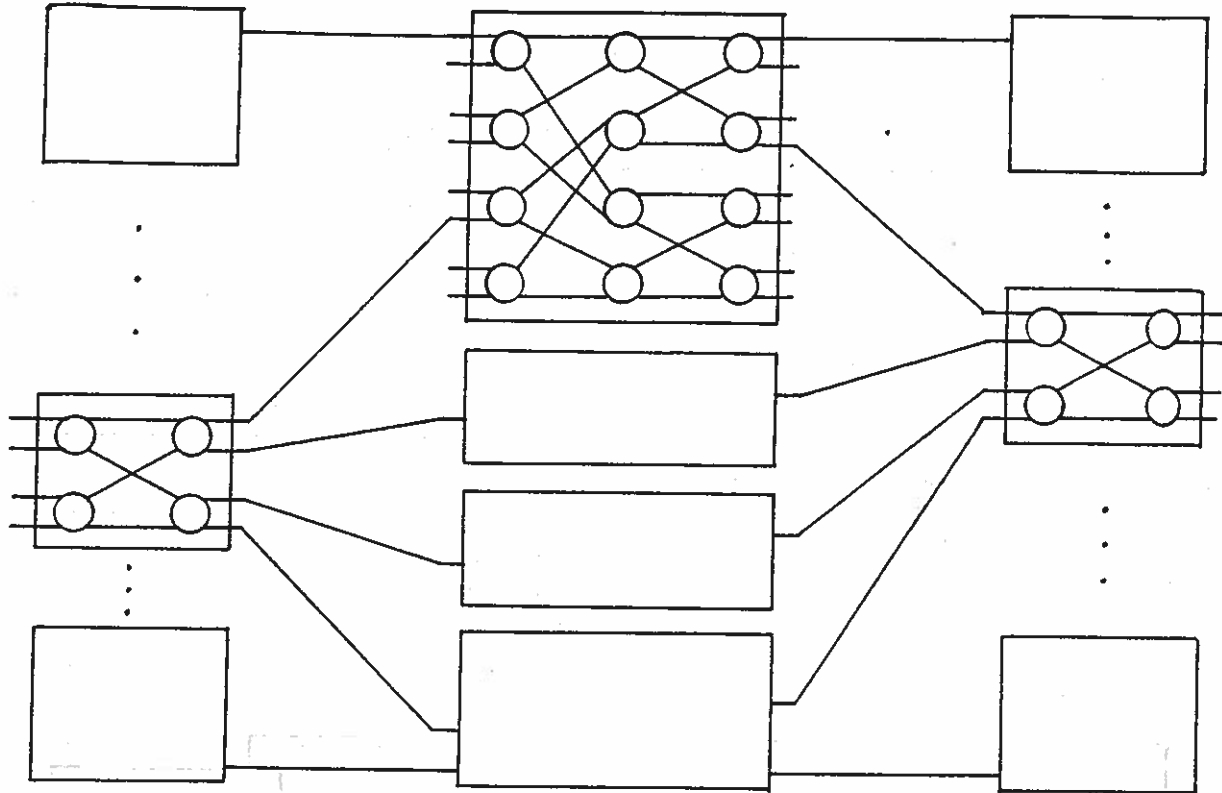


Figure 3.3: Construction of Delta Network with Distribution Stages

loading, it never makes sense to have $k + d$ an even number, since a network with one fewer distribution stage has the same worst-case loading characteristics. We note that this result can be readily generalized to networks with nodes having m input and output ports. The bound in the statement of the theorem becomes $nm^{-\lceil(k+d)/2\rceil}$ (with $k = \log_m n$).

In [37], Lea proposes a variant of the delta network that we refer to as the *alternate routing network*. We can define this network recursively using the composition operation. The base network is denoted by A_1 and consists of four input ports and four output ports connected to a single internal node. For $i > 1$, $A_i = A_1 \textcircled{2} A_{i-1}$. An example of an alternate routing network is given in Figure 3.4. Note that an alternate routing network with k stages has $n = 2^{k+1}$ ports. Given any connection (x, y, ρ) , if u is in the first $k - 1$ stages and lies on some path from x to y , then two of u 's output links lie on paths from x to y . Consequently, whatever load is induced on the input links of u will be shared by two of u 's output links. The following theorem characterizes the worst-case loading of an alternate routing fabric. We note that essentially the same result is stated (in somewhat different terms) in [37].

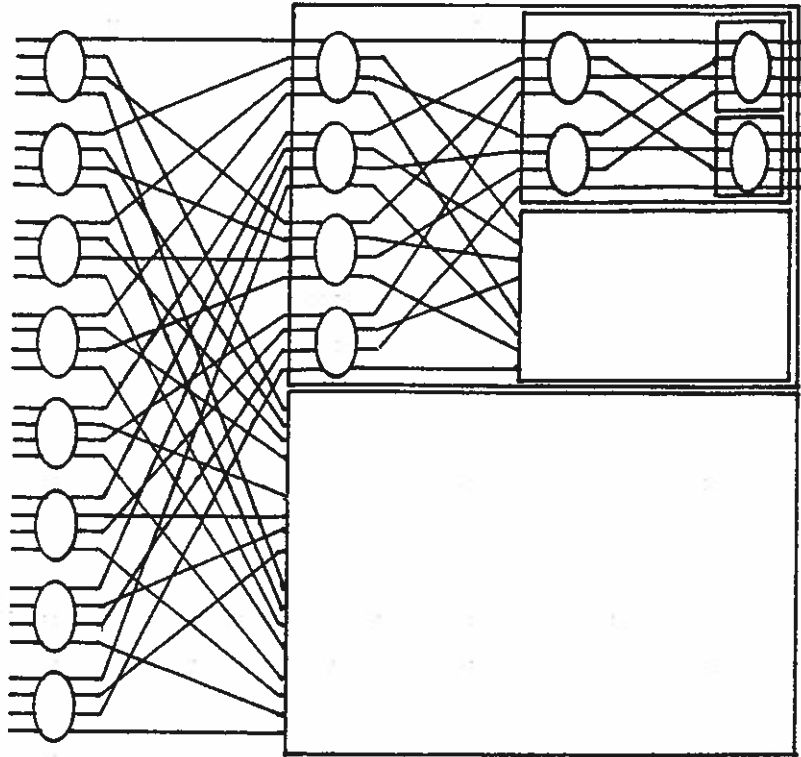


Figure 3.4: Construction of Alternate Routing Network

THEOREM 3.1.3. *Let $C = \{c_1, \dots, c_r\}$ be an α -bounded configuration for A_k . Then $\lambda_\ell(C) \leq \alpha n^{1/3}$ for all links ℓ .*

Loading in Copy Networks

The broadcast packet switch of [59] is one of several proposed systems for multi-point communication. In this section we study the worst-case loading of the copy network, which gives that system the ability to handle multipoint communication. We also consider several variants.

When dealing with copy networks, we must modify our definition of connection. In the current context, we define a connection to be an ordered triple (x, F, ρ) , where x is the input port of the copy network where packets belonging to the connection enter, F is the *fanout* of the connection and ρ is the load induced by the connection at the input port x . The fanout of the connection is the number of copies that must be produced by the copy network for each input packet. We say that a traffic configuration C is α -bounded if $\lambda_\ell(C) \leq \alpha$ for all input ports ℓ

and $\sum_{(x,F,\rho) \in C} \rho F \leq \alpha n$, where n is the number of input and output ports. A legal configuration is one that is 1-bounded.

Reference [59], describes a copy network that is topologically identical to a delta network. However, the nodes of a copy network replicate received packets under certain conditions. Specifically, a node may replicate a packet if the number of output ports reachable from that node is less than $2F$, where F is the fanout of the connection the packet belongs to. Packets that are not replicated are routed to an arbitrarily selected output. Hence, if α is the load induced on the input links of a node u by a connection (x, F, ρ) , then the load induced on each of u 's output links is α , if the number of output ports that can be reached from u is $< 2F$ and $\alpha/2$ otherwise.

Given these definitions, we find that for a connection $c = (x, F, \rho)$ and a link ℓ in stage i ,

$$\lambda_{\ell}(c) = \begin{cases} 0 & \text{if there is no path from input } x \text{ to link } \ell \\ \rho 2^{-i} & \text{if there is a path and } 0 \leq i \leq k - \lceil \log_2 F \rceil \\ \rho 2^{-(k - \lceil \log_2 F \rceil)} & \text{if there is a path and } i \geq k - \lceil \log_2 F \rceil \end{cases}$$

Our first theorem, which was first proved in [5] shows that the worst-case loading in a copy network is bounded.

THEOREM 3.1.4. *Let $C = (c_1, \dots, c_r)$ be any α -bounded configuration for an n -port copy network. Then, $\lambda_{\ell}(C) \leq 3\alpha$ for all links ℓ .*

There exist legal traffic patterns approaching the bound in Theorem 3.1.4. Copy networks can also be constructed using nodes with $m > 2$ input and output ports. In such networks, a node replicates a packet m times if the number of reachable output ports is less than mF . Surprisingly, the worst-case performance of such a copy network is worse than for a copy network constructed from binary nodes.

THEOREM 3.1.5. *Let $C = (c_1, \dots, c_h)$ be an α -bounded configuration for an n -port copy network constructed from m -port nodes. Then $\lambda_{\ell}(C) < \alpha(m + 1)$, for all links ℓ .*

The proof of this is very similar to that of Theorem 3.1.4. Again, the bound is the best possible; there exist legal traffic configurations that induce loads approaching $m + 1$ on some internal links.

As with routing networks, we can improve the worst-case performance of a copy network by adding distribution stages. The topology of such a network is identical to a routing network with added distribution stages. The effect on the worst-case loading is captured by the following theorem.

THEOREM 3.1.6. *Let $C = (c_1, \dots, c_h)$ be an α -bounded configuration for a copy network with k copy stages and d distribution stages. Then $\lambda_\ell(C) < \alpha(1 + 2^{1-d})$, for all links ℓ in stages 0 to $k + d - 1$; $\lambda_\ell < 2\alpha$ for all links ℓ in stage $k + d$.*

Theorem 3.1.6 shows that the worst-case loading in a copy network can be brought very close to α in all but the last stage links, by adding a few distribution stages. We note that Theorem 3.1.6 can be generalized to copy networks constructed with m -ary nodes. In this case the bound on the worst-case loading becomes $\alpha(1 + m^{1-d})$ for all but the last stage and m for the last stage.

3.2. Generalized Non-Blocking Networks

In this section we introduce a generalization of the classical theory of non-blocking networks. This new theory applies to packet switching fabrics in which all packets in a given connection are constrained to follow the same path. In the broadcast packet switch architecture this is not the case within a switch module but is the case for a large switching system comprising many switch modules.

We start with some definitions. We define a network as a directed graph $G = (V, E)$ with a set of distinguished input nodes I and output nodes O , where input nodes have one outgoing edge and output nodes one incoming edge. A *connection* in a fabric is a triple (x, y, ω) where $x \in I$, $y \in O$ and $0 \leq \omega \leq 1$. We refer to ω as the *weight* of the connection and it represents the bandwidth required by the connection. A *route* is a path joining an input node to an output node, with intermediate nodes in $V - (I \cup O)$. A route r *realizes* a connection (x, y) , if x and y are the input and output nodes joined by r .

A set of connections is said to be *compatible* if for all nodes $x \in I \cup O$, the sum of the weights of all connections involving x is ≤ 1 . Similarly a collection of routes is compatible if for all edges $(x, y) \in E$ the sum of the weights of all routes involving (x, y) is ≤ 1 . A set of connections is said to be *realizable* if there is a set of compatible routes that realizes that set of connections.

A fabric is said to be *rearrangeably nonblocking* (or simply *rearrangeable*) if for every set C of compatible connections, there exists a set R of compatible routes that realizes C . A fabric is *strictly nonblocking* if for every set of compatible routes R , realizing a set of connections C , and every connection c compatible with C , there exists a route r that realizes c and is compatible with R . For strictly nonblocking fabrics, one can choose routes arbitrarily and always be guaranteed that any new connections can be satisfied without rearrangements. Other fabrics can be nonblocking if one chooses routes judiciously. Such fabrics are said to be *wide-sense nonblocking*.

Sometimes, improved performance can be obtained by placing constraints on the traffic imposed on a network. We will consider two such constraints. First, we limit the weight associated with any connection and denote this maximum weight by B . We also limit the sum of the weights of connections involving a node x in $I \cup O$ to β . Clearly $0 \leq B \leq \beta \leq 1$. We say a network is strictly non-blocking for particular values of B and β if for all sets of connections for which the maximum connection weight is B and total port weight is β , the network cannot block. The definitions of rearrangeably non-blocking and wide-sense non-blocking networks are extended similarly. (Note that such theoretical restrictions may not restrict practical systems significantly, but may rather imply greater capacity in the internal links of a network than in the external links connecting different networks together. Such a *speed advantage* is commonly used in high speed systems.)

We start with an elementary observation to illustrate the definitions. A binary delta network with n ports is strictly non-blocking if $\beta \leq \sqrt{1/n}$. Furthermore, if $\log_2 n$ is odd, it suffices for $\beta \leq \sqrt{2/n}$. This is similar to the first theorem in the previous section. We now list a few more substantive theorems.

THEOREM 3.2.1. *Let c be a fixed positive integer. Any network constructed from nodes with at most c incident arcs, that is non-blocking for $\beta = 1$ has $\Omega(n^2)$ nodes.*

This implies that for unrestricted input traffic ($\beta = 1$) a crossbar is essentially a best possible strictly non-blocking network. This contrasts sharply with the situation for space division networks. Our next theorem captures a general trade-off between the complexity of a network and the maximum port weight β .

THEOREM 3.2.2. *Let $c \geq 3$ be an integer, $0 \leq \beta \leq \alpha \leq 1$. Let F_1, F_2, \dots be a family of networks in which each node is incident to at most c edges and the number of nodes in F_n is $O(f(n))$. If for all n , F_n is strictly (rearrangeably, wide-sense) non-blocking when the maximum port weight is α , then there exists a family of networks F'_1, F'_2, \dots where for each n , F'_n is an n port network with $O(n + f(\beta n/\alpha))$ nodes and F'_n is strictly (rearrangeably, wide-sense) non-blocking when the maximum port weight is β .*

The networks in F' are constructed by selecting an appropriate network in F and adding a set of $(\alpha/\beta) : 1$ concentrators at the front and a set of $1 : (\alpha/\beta)$ expanders at the back.

A binary Benes network is the network $D_{k,k-1}$ defined in the previous section where $k = \log_2 n$. A Cantor network of multiplicity m consists of a set of $1 : m$ expanders feeding into m parallel Benes networks and followed by a set of $m : 1$

concentrators. The complexity of a Benes network is $O(n \log n)$ and the complexity of the Cantor network is $O(mn \log n)$.

THEOREM 3.2.3. *A Cantor network of multiplicity m is strictly non-blocking if $m \geq \frac{\beta}{1-\beta} \log_2 n$.*

COROLLARY 3.2.1. *The Benes network is strictly non-blocking if $\frac{1-\beta}{\beta} \geq \log_2 n$.*

Theorem 3.2.3 is similar to a corresponding result for space-division networks, which states that a multiplicity of $m = \log_2 n$ is sufficient to ensure non-blocking operation. Corollary 3.2.1 implies that for $\beta \leq 1/(1 + \log_2 n)$, the Benes network is strictly non-blocking. Moreover, by applying Theorem 3.2.2 to this result, we observe that there exists a strictly non-blocking network with $O(n)$ complexity and $\beta = 1/(\log_2 n)^2$. The Benes network is rearrangeably non-blocking in the space division case. It turns out that it is also rearrangeably non-blocking for varying rate connections in certain restricted cases.

THEOREM 3.2.4. *For $B \leq 4/(4 + \log_2 n)$, the Benes network is rearrangeably non-blocking.*

THEOREM 3.2.5. *Let r be any positive integer. The Benes network is rearrangeably non-blocking for sets of connections in which all weights ω satisfy $1/(r+1) < \omega \leq 1/r$.*

We can show that the Benes network is not rearrangeably non-blocking for unrestricted traffic configurations, but we conjecture that for small values of m , the Cantor network of multiplicity m is. Theorems 3.2.3 and 3.2.5 suggest the possibility of a network in which connections with large weights are separated and handled in a rearrangeably non-blocking fashion. This observation leads to the following theorem.

THEOREM 3.2.6. *Let $r \geq 2$ be an integer and $m \geq (r-1) + \frac{r}{r-1} \log_2 n$. A Cantor network of multiplicity m is rearrangeably non-blocking for connections of weight $> 1/r$ and wide-sense non-blocking for connections of weight $\leq 1/r$.*

We have also considered generalizations of Clos networks. Let $C_{N,n,m}$ denote a three stage Clos network with N input and output ports and constructed from $n \times m$ crossbars in the first stage, $(N/n) \times (N/n)$ crossbars in the middle stage and $m \times n$ crossbars in the third stage. Also, let b be the *minimum* weight for a connection.

THEOREM 3.2.7. *The Clos network $C_{N,n,m}$ is strictly non-blocking if*

$$m > 2 \left[\max_{b \leq \omega \leq B} \min \left\{ \left\lfloor \frac{\beta n - \omega}{1 - \omega} \right\rfloor, \left\lfloor \frac{\beta n - \omega}{b} \right\rfloor \right\} \right]$$

This yields several immediate consequences. If we let $b = B = \beta = 1$, the effect is to operate the network in a space-division mode and the theorem states that we get non-blocking operation when $m > 2n - 1$ as is well-known. If we let $b = 0$ and $B = \beta = 1/2$, we also get non-blocking operation for $m = 2n - 1$.

We have only scratched the surface of this research topic. There are several directions in which this work may be extended. While we have good constructions for strictly non-blocking networks, we expect our results for rearrangeably non-blocking networks can be considerably improved. Another interesting topic is non-blocking networks for multipoint connections. While this has been considered for space-division networks, it has not been studied for networks supporting connections of any size. Another area to consider is analysis of blocking probability in such networks. We expect blocking probability to be highly dependent on particular choices of routing algorithm and anticipate that extensive simulation will be required to explore this in depth.

3.3. Packet Misordering

This section presents some initial simulation results which attempt to assess the likelihood that packets passing through a broadcast packet switch become misordered. The results presented here are for a configuration consisting of a copy network, distribution network and routing network, all with 64 input and output ports and all comprising binary switch elements with two buffer slots per input.

Our results are summarized in the two plots shown in Figure 3.5. The plot on the left gives the distribution of the delay incurred by packets passing through a switch fabric. Note that for an offered load of $\rho = 0.4$ (the maximum allowed under normal operating conditions) the vast majority of the packets pass through the switch fabric in under ten packet times and at this loading level, only about one packet in 10^5 is delayed as much as 20 packet times. Since the switch fabric has a 2:1 speed advantage over the external links, packets that arrive 10 packet times apart on an external link are very unlikely to get misordered by the switch fabric. This in turn suggests that resequencing packets on the output side of the switch fabric may be viable using only a small resequencing buffer. In particular, one could time-stamp packets on receipt from a link, then order them by time-stamp

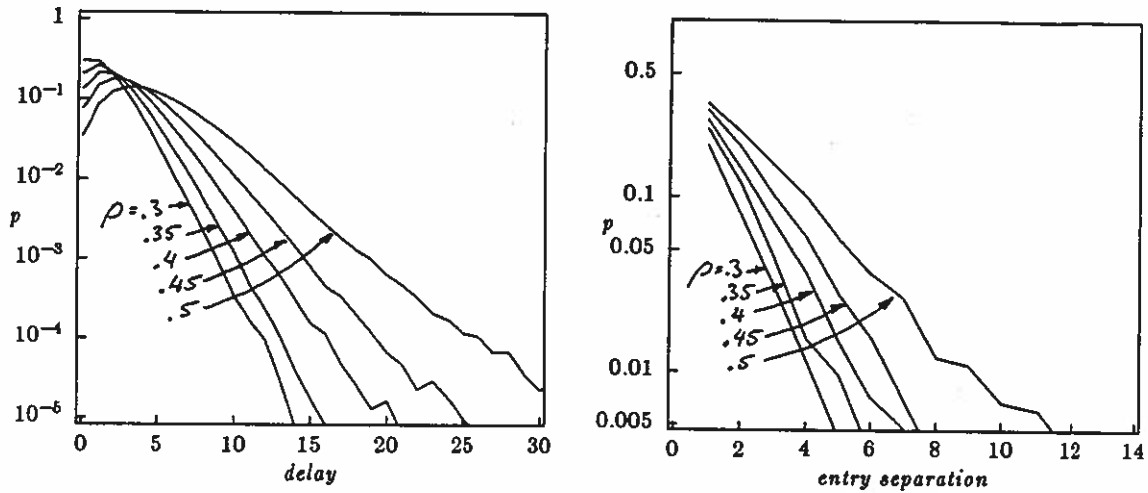


Figure 3.5: Distribution of Delay and Misordering Probabilities

in the transmit buffer. If in addition, one never transmits a packet that isn't at least say 10 packet times old, the likelihood of misordering can be reduced to a very small level. While the logic to put packets in time-stamp order adds some complexity to the output buffer, the incremental cost is fairly small.

The plot on the right examines, the likelihood of packets being misordered more closely. In the simulations on which these results are based, all packets entering the copy network on input port 0 were treated as belonging to a point-to-point connection going to output port 0 of the routing network. The plot shows the fraction of all pairs of packets entering the copy network with a separation of x packet times that were misordered. So for example, for $\rho = 0.4$, about 1% of the pairs arriving six time units apart were misordered. If one extrapolates from these curves, one finds that less than one pair of packets in 10^6 that arrive with a separation of 20 packet times are misordered.

4. Prototype Hardware Design

Faculty
Research Associate
Graduate Students

Jonathan Turner
Pierre Costa
Neil Barrett
Shabbir Khakoo
George Robbert
James Sterbenz
Einir Valdimarssen

A prototype of a BPN switch module is being designed. The purpose of this prototyping effort is to provide a convincing demonstration of feasibility, allow detailed examination of implementation issues and provide a testbed for future experimental efforts at higher levels. During the past year, we have designed several trial chips in order to obtain a detailed understanding of implementation issues and to deepen our experience with the design process. We are now designing integrated circuits to be used in our prototype system. These will incorporate a number of fundamental improvements based on our experience with the trial chips and related performance studies.

The overall structure of the prototype packet switch is shown in Figure 4.1. The *Connection Processor* (CP), shown at the top of the figure, is a general purpose computer that provides overall control of the system, including connection establishment. The heart of the system is an eight port *Switch Fabric* (SF) comprising a *Copy Network* (CN), a set of *Broadcast Translation Circuits* (BTC) and a *Routing Network* (RN). A set of *Packet Processors* (PP) provide the interface between the SF and the high speed *Fiber Optic Links* (FOL) that are used to interconnect different switches. The CP communicates with the rest of the system through the *CP Interface* (CPI). The system is operated in a highly synchronous fashion, with global timing provided by the single timing circuit shown at the top of the figure. The prototype system is designed to support FOL speeds of at least 150 Mb/s. To achieve this, the internal data paths are required to operate at an effective rate of at least 300 Mb/s. Consequently, the prototype will use eight bit wide data paths and the target clock speed is 50 Mb/s.

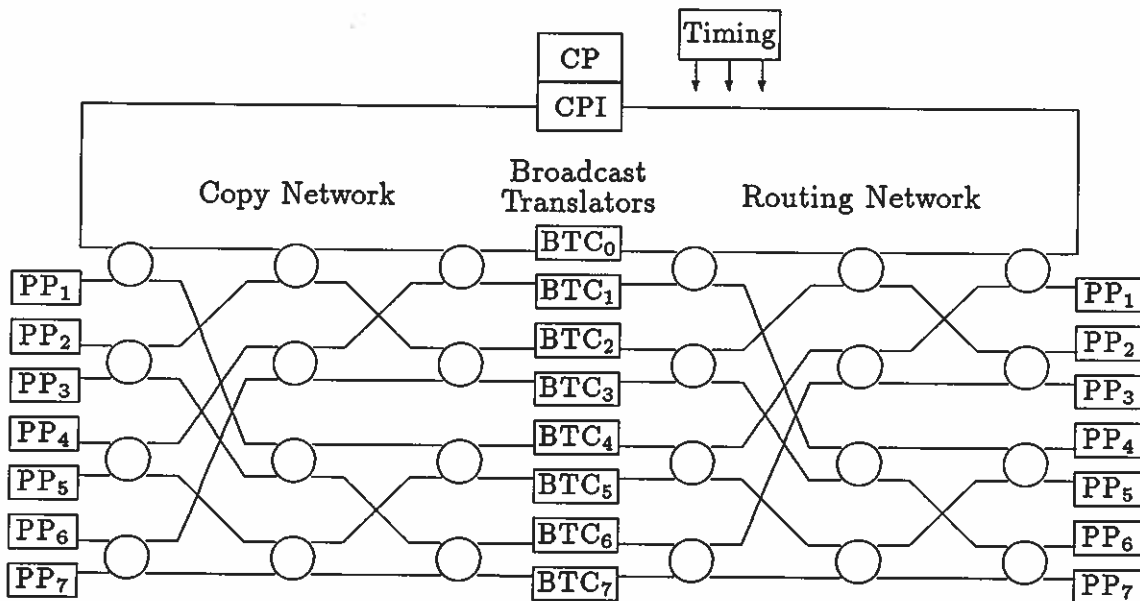


Figure 4.1: Prototype Switch Module

Custom integrated circuits are being designed for the switch elements, BTCs and Packet Processors (PP). The BTC and switch element designs will require one chip apiece, the PP design will require two or three chips. A total of approximately 50 custom chips are required to implement the prototype switch module. At this time, trial designs have been completed for the BTCs and switch elements. These are being fabricated currently and will be tested on return. We are currently revising these designs, in part to meet the speed objectives for the prototype, and at the same time are developing detailed designs for the PP. Design of the CP interface and timing circuitry is also underway. The remainder of this chapter describes the format of the packets used within the prototype and gives a top level description of the design of the various components.

4.1. Packet Formats

This section describes the formats of packets used in the switch. There are two primary packet formats: external and internal. Packets are carried in external format on the fiber optic links connecting switches, and in internal format within each switch. The PP translates between these two formats. Note that higher level processes may define additional packet formats; this section details only those fields that are of direct concern to the prototype hardware.

External Packet Format.

Each external packet is organized as a sequence of 8 bit wide words. Each packet contains exactly 76 words, the first 3 of which constitute the packet *header*. The last word of the packet is used for a frame checksum. When transmitted on the external transmission links, external packets are separated by a SYNC pattern that allows the receiver to identify packet boundaries. The meanings of the external fields are given below.

- *Packet Type* (PTYP). Identifies one of several types of packets, including ordinary data packet, control packets and test packets.
- *External Logical Channel Number* (ELCN). Logical channel numbers are used to identify which connection a packet belongs to. For the prototype, only 256 distinct logical channels are recognized.
- *Information* (I). Normally contains user information. In the case of control packets, may contain additional control information. Individual words are denoted $I[0], I[1], I[2], \dots$ with $I[0]$ being the first word of the I field.
- *Frame Check* (FC). The frame check is used to detect errors in the packet. A simple check sum over the first 75 bytes of the packets is used.

Internal Packet Format

Each internal packet is organized as a sequence of nine bit wide words, including an odd parity bit. Each packet contains exactly 80 words, the first five of which constitute the packet *header*. The structure of the packet is shown in Figure 4.2. The meanings of the fields are given below.

- *Routing Control* (RC). This field determines how the packet is processed by the switch elements. The possible interpretations are listed below.
 - 0 *Empty Packet Slot*
 - 1 *Point-to-Point Data Packet*
 - 2 *Broadcast Packet*
 - 4 *Specific-Path Packet*
- *Operation* (OP) This field specifies which of several control operations is to be performed for this packet. The possible values of the field and the corresponding functions are listed below.

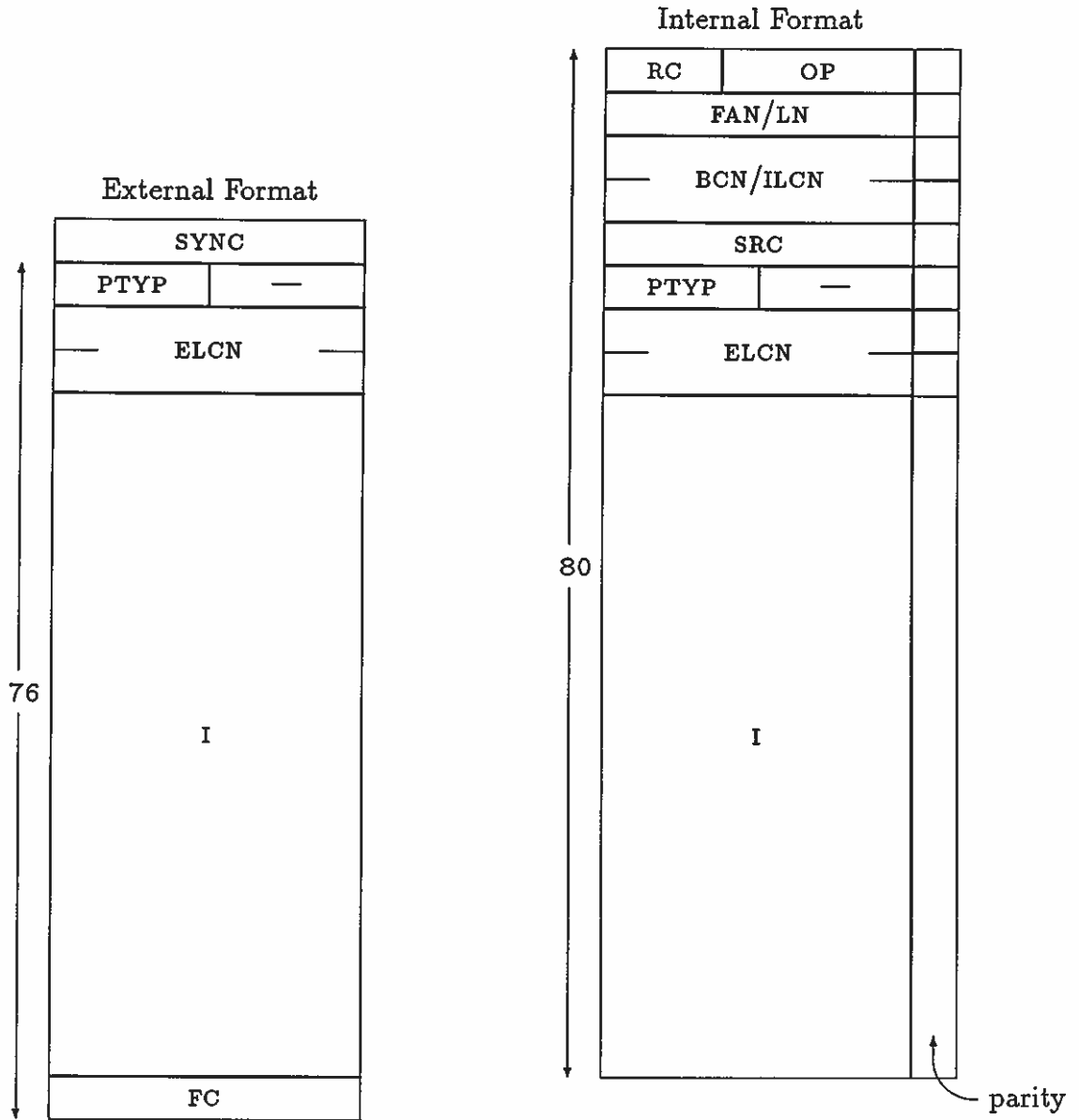


Figure 4.2: Packet Formats

- 0 *Vanilla Packet*. No control functions.
- 1 *Read LCXT Block*. Directs PP to read a block of 16 entries from the Logical Channel Translation Table. I[0] specifies which block to read. The data is copied into I[1]–I[64] and the packet is returned to the CP.
- 2 *Write LCXT Block*. Directs PP to write a block of 16 entries to the Logical Channel Translation Table. I[0] specifies the block to write. The data to be written is in I[1]–I[64].
- 3 *Read PP Parameter Block*. Read the contents of the PP parameter block into I[1]–I[64] of the packet and return the packet to the CP.
- 4 *Write PP Parameter Block*. Write the contents of I[1]–I[64] into the PP parameter block.
- 5 *Switch Test Packet*. When received by a PP is returned to the SF with a new routing field. The new routing field is obtained by rotating the entire contents of the packet by five words.
- 6 *Read BTT Block*. Directs BTC to read and return a block of 16 entries from the Broadcast Translation Table (BTT). I[0] field specifies which block to read. The data is copied into I[1]–I[64] and the packet returned to the CP.
- 7 *Write BTT Block*. Directs BTC to write information into a block of 16 entries of the BTT. I[0] field specifies which block to write. The data to be written is in I[1]–I[64].
- 8 *BTT Single Entry Update*. Directs all BTCs in a given group to update an entry in their BTTs. BCN gives the broadcast channel number of the connection, I[0] is the new fanout, I[1] is the block of Broadcast Copy Indices that are to be updated (16 BCIs to a block) and I[2]–I[65] contains the block of 16 entries. Each BTC calculates its broadcast copy index, j , for the connection and if $\lfloor j/16 \rfloor$ equals the block number in I[1], it copies I[4($j \bmod 16$)]–I[($j \bmod 16$) + 3] to BTT[BCN].
- 9 *Read BCIT Block*. Directs the BTC to read the contents of one block of 64 BCIT entries into I[1]–I[64] and return the packet to the CP. I[0] specifies the block to read.
- A *Write BCIT Block*. Directs the BTC to write the information in words I[1]–I[64] into one block of the BCIT. I[0] specifies the block to write.

C–FF *Reserved*.

- *Destination (DST)*. The interpretation of these three words depend on the value of RC.

- *Fanout (FAN)*. If $RC = Broadcast\ Packet$, the second word of the packet is taken to be the fanout, that is the number of switch fabric output ports that require copies of the packet.
 - *Broadcast Channel Number (BCN)*. If $RC = Broadcast\ Packet$, the third and fourth words of the packet are taken to be the broadcast channel number. All packets within a particular multi-point channel have the same broadcast channel number. Only 256 distinct BCNs are recognized.
 - *Link Number (LN)*. If $RC = Point-to-Point\ Packet$, the second word of the packet is taken to be the number of the outgoing link to which the packet should be delivered.
 - *Internal Logical Channel Number (ILCN)*. If $RC = Point-to-Point\ Packet$, the third and fourth words of the packet are taken to be the internal logical channel number. This will become the external logical channel number when the packet exits the switch module.
 - *Specific Path Specification*. If $RC = Specific-Path\ Packet$, the three words of the DST field specify output ports for each of the three networks. The packet will be routed through each of these.
- *Source (SRC)*. The number of the most recent PP through which the packet has passed. For vanilla packets, this will be the number of the link on which the packet entered the switch. For test packets it will be changed as the packet passes through different PPs.

4.2. Timing

The system is operated in a highly synchronous fashion. All packets are the same length and pass through the switch fabric in synchrony with one another. There is a global packet cycle that determines the timing of all events within the system. Incoming packets are received by the packet processors and synchronized to this packet cycle. Each cycle is referred to as an *epoch*. The length of an epoch is 84 clock times or $1.68\ \mu s$. This allows time for one packet to be processed and leaves a guard time of four clock periods between packets.

The global timing generator provides the base 50 MHz clock that drives the system plus a set of signals that define various instants within the global time reference. The notation gt_i is used to denote clock cycle i in the global time reference. By definition, gt_0 is the time at which packets start to enter the leftmost stage of the copy network. The nodes of the switch fabric delay packets passing through them for exactly 32 clock times and the BTC delays packets for exactly

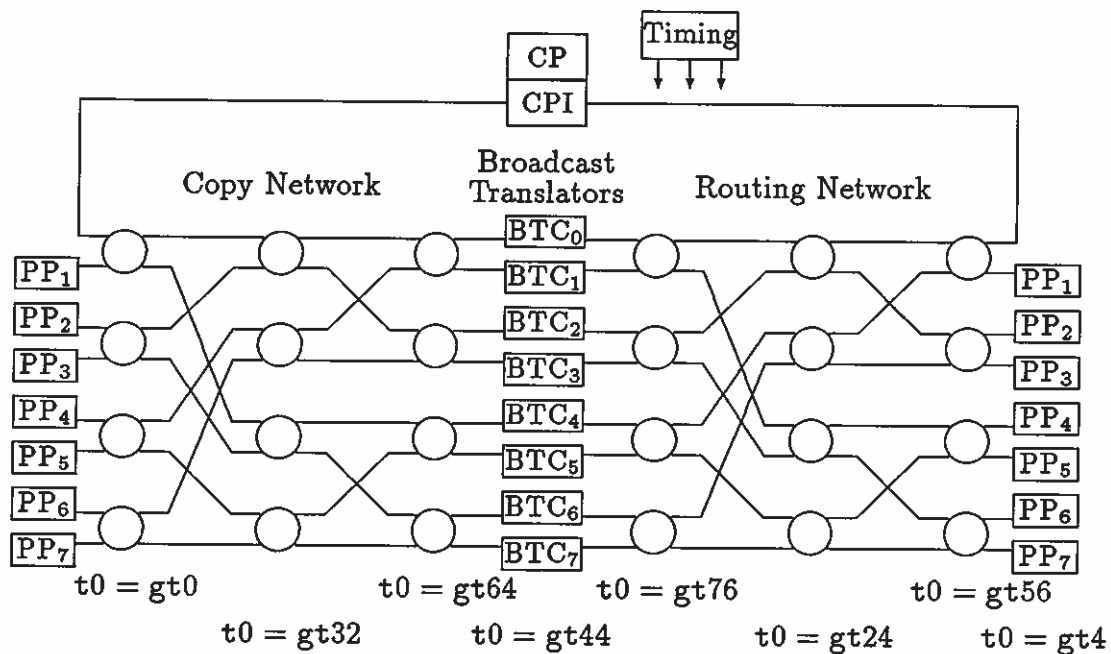


Figure 4.3: Local and Global Timing Relationships

64 clock times. Consequently, packets pass from the leftmost stage of the copy network to the next stage at $gt32$ and so forth.

Every component in the system has a local time reference which is typically synchronized to the point in the global time reference at which that component can start to receive a packet on one of its input links. The notation t_0 denotes the starting point of the epoch for a particular component's local time reference. Each of these local time references is synchronized to the global time reference as shown in Figure 4.3.

4.3. Packet Switch Element

The Packet Switch Element chip (PSE) is the 2×2 VLSI switch element used in the binary routing, copy and distribution networks. The PSE directs packets to one or both outputs based on packet type (point-to-point, broadcast, or specific-path), switch operation mode (routing, copy, or distribution), and the contents of the LN/FAN field.

The prototype version of the PSE differs from the initial trial chip in several important respects. The objective of these changes has been to eliminate constraints

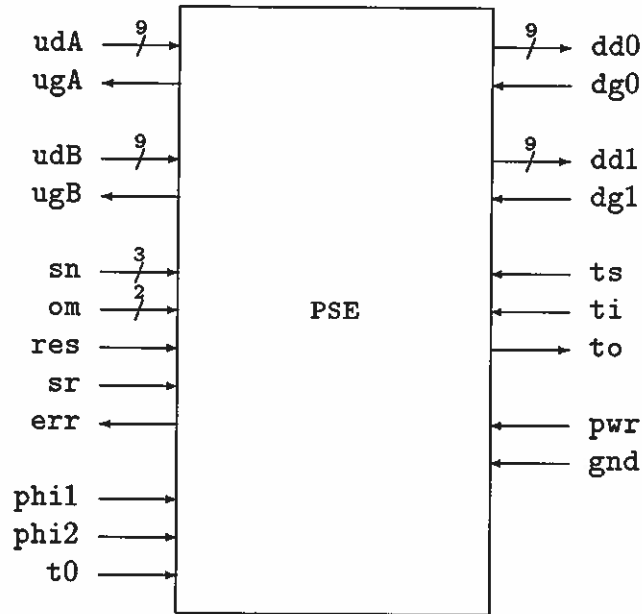


Figure 4.4: External Interface for Packet Switch Element Chip

on the speed of operation of the PSE. The design outlined below is expected to run at clock speeds of 50 Mb/s, as opposed to 10 Mb/s for the trial chip. This improvement is due largely to changes in some basic design decisions. The most important change is to modify the way in which grant propagation is handled. In the system as described in [59], grants are propagated from the output of the routing network back through the inputs to the copy network before packets can flow forward. This design makes best use of the nodes' internal buffers but places tight constraints on the number of clock cycles a node can delay a packet. In the new design a node makes decisions on its upstream grants independent of the status of the downstream grants. This change greatly relaxes the constraint on the number of clock cycles a node can delay a packet, which in turn makes it possible to increase the speed of the clock. Because this change reduces the effectiveness of node buffers, we have also decided to switch from a design with a single buffer per input to one with two buffers per input.

External Interface

The external leads of the switch element are shown in Figure 4.4 and described briefly below.

- *Upstream data leads* (udA,udB). Incoming data from upstream neighbors. Nine bits wide, including parity.
- *Upstream grants* (ugA,ugB). Grant signals to upstream neighbors. When asserted, grants permission to transmit packet on corresponding data leads during subsequent epoch.
- *Downstream data leads* (dd0,dd1). Outgoing data to downstream neighbors. Nine bits wide, including parity.
- *Downstream grants* (dg0,dg1). Grant signals from downstream neighbors. When asserted, grants permission to transmit packet on corresponding data leads during subsequent epoch.
- *Stage number* (sn). Three bit stage number. Each network has up to eight stages (columns), numbered from 0, with stage 0 being the last (rightmost) stage in a network.
- *Operating mode* (om). Two bit code identifying which of three possible operating modes the switch element implements. 1 for route, 2 for distribute, 3 for copy.
- *Reset* (res). Initialize all internal control registers; this causes any packets in the node to be discarded.
- *Soft Reset* (sr). Clear the error flag.
- *Error* (err). Report parity violation or other error.
- *Clock* (phi1,phi2). Two-phase, non-overlapping clock.
- *Start of Packet Cycle* (t0). Goes high when first word of packet is present on ud leads.
- *Test Shift* (ts). Shift lead for controlling shifting of test data.
- *Test In* (ti). Input lead for test data.
- *Test Out* (to). Output lead for test data.
- *Power* (pwr).
- *Ground* (gnd).

Global Operation

A single PSE circuit is used to implement the routing, copy and distribution networks. Packets are handled based on the information in the packet headers and either forwarded to the appropriate output (or outputs) or held until the required output(s) is available. The grant signals are used by nodes to control the arrival of packets from their upstream neighbors. In general, a node asserts a grant, allowing a new packet to arrive if it has an available buffer in which to store the packet. Each node can store up to four complete packets in its internal buffers.

PSE routing decisions are based on the operation mode and RC field, as specified below.

- For $om = \text{route}$; use bit sn of the LN field to select an output port, where sn is the stage number.
- For $om = \text{copy}$; if RC is broadcast, and FAN exceeds 2^{sn} , where sn is the stage number, send copies of packets to both output ports. If RC is specific-path, use bit sn of LN field to select an output port. Otherwise, distribute.
- For $om = \text{distribute}$; if RC is specific-path, use bit sn of LN field to select an output port. Otherwise, distribute.

When arbitrary routing choices can be made, the following policies are used to make decisions:

- Ties among input ports for a given output port are arbitrarily broken based on the last input port favored, to avoid individual starvation.
- Packets that can proceed to either output are uniformly and arbitrarily distributed (all packets in distribution network, point-to-point packets and broadcast packets not replicated in copy network).
- Packets requesting both outputs in the copy network are favored over packets requiring only one.
- Packets requesting a specific output are favored over packets which can use either.

The clock period during which the first word of a packet appears on the upstream data leads is called t_0 and in general, the clock period during which word i appears is called t_i . The delay through a node is 32 clock times, or 640 ns. So, if an incoming packet can be switched through a node without buffering, the first

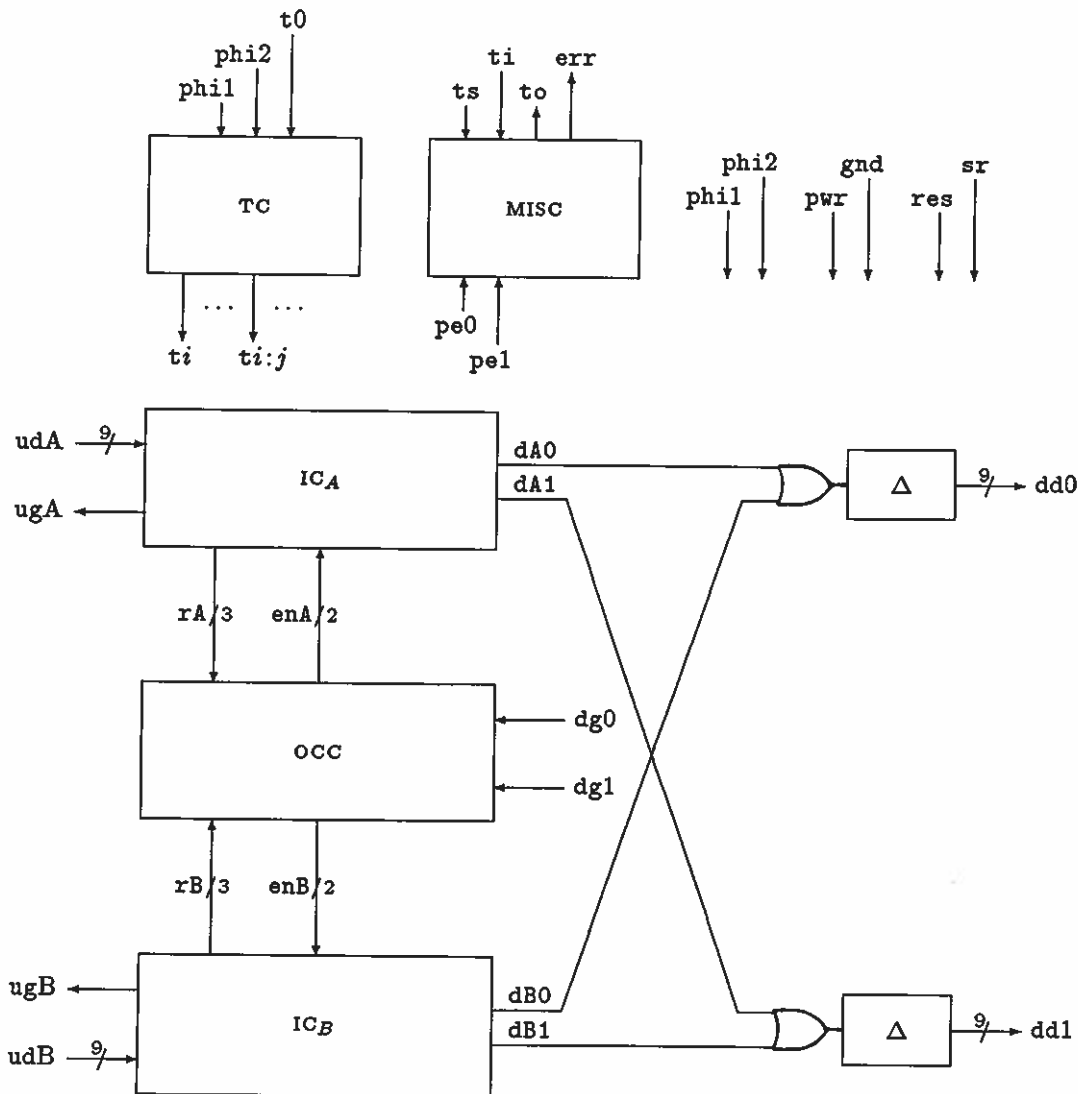


Figure 4.5: Block Diagram of Packet Switch Element Chip

byte will appear on the output at t_{32} . Each node makes its upstream grant signals available at t_{32} in the node's frame of reference and holds the grant leads in that state until t_{32} of the subsequent cycle. Consequently, the grant signal is available to the upstream neighbor any time after t_{64} in the neighbor's frame of reference.

Internal Components

A block diagram of the PSE appears in Figure 4.5. The major components are described below.

- *Output Control Circuit (OCC)*. The OCC arbitrates access to the two output ports, based on the downstream grant signals and port requests received from the input circuits. The port requests are given in the form of three bit request vectors, rA and rB ; a value of 101 requests access to a output port 0, 110 requests output port 1, 111 requests both output ports and 100 requests a single output port, with either one being acceptable. The individual bits of these three bit codes are assigned the names r_n , r_1 , and r_0 with the suffix A or B included when necessary to designate a specific side. The response is given in the form of two bit enable vectors enA and enB ; a value of 01 grants access to port 0, a value of 10 grants access to port 1 and a value of 11 grants access to both. The individual bits have the names $en1$ and $en0$.
- *Input Circuits (ICA,ICB)*. There is one input circuit for each input port. Each IC includes two buffers large enough to hold a single packet, plus control circuitry to extract information from the packet header, generate the request vector for the OCC and use the resulting enable vector to make decisions on the disposition of the packet. It also modifies the packet header when necessary.
- *Timing Circuit (TC)*. This circuit generates signals of the form t_i and $t_i:j$, for various values of i, j . Signal t_i is high during clock period t_i of the epoch; in particular it goes high during ϕ_{i2} of the preceding clock cycle and goes low before ϕ_{i2} goes high again. Signal $t_i:j$ is similar; it is high during t_i and stays high through t_j .

Output Control Circuit. The Output Control Circuit is a PLA with ten inputs and six outputs plus two flip flops which store the values of a pair of tie-breaker variables. The flip flop u_i specifies the input port that was most recently favored the last time a tie was broken; in particular, if input port A was most recently favored u_i is 0, otherwise it is 1. Similarly, u_o gives the number of the output port that was most recently used during an epoch when only one output port was used.

Input Circuit. The structure of the input circuit is shown in Figure 4.6. The main blocks are summarized below.

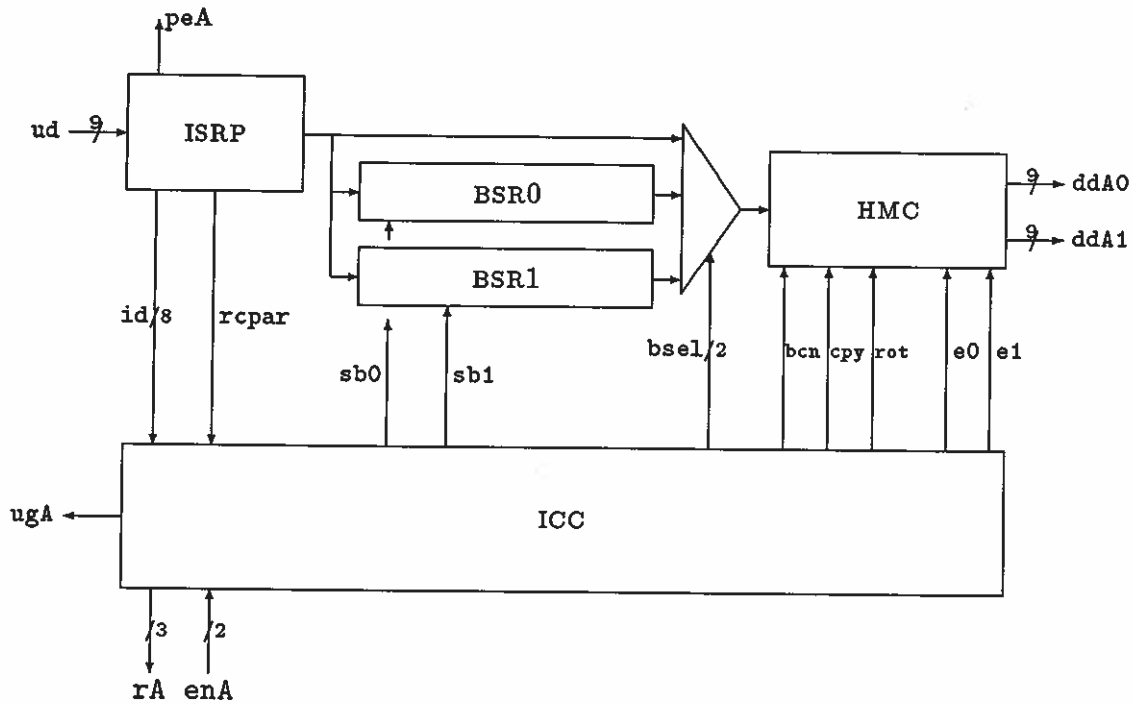


Figure 4.6: Input Circuit

- *Input Shift Register (ISR)*. The input shift register is a 20 stage static shift register with an output tap after the first stage and a parity checker. Packets are shifted into the ISR from the upstream data lines. The first stage of shift delay provides synchronization. The remaining stages allow time for control and routing decisions to be made by the input and output control circuits. The leads hd are connected to the output of the first shift register stage (data bits only) and provide access to the header information. The signal $rcpar$ is 1 if the parity of the first byte of the packet is incorrect. This is used to suppress copying of packets with incorrect routing control.
- *Input Control Circuit (ICC)*. The ICC controls the flow of packets through the input circuit. It extracts and decodes header information from incoming packets and stores the decoded information for packets stored in the packet buffers. Using this information, it requests output ports from the OCC and based on the results, controls the flow of packets through the IC. It also generates the upstream grant signals. A more detailed description of the ICC appears below.
- *Buffer Shift Register (BSR)*. Each BSR is a static 80 stage shift register, with the shift control provided by the ICC. A total of two BSRs are provided. The

buffers are followed by a multiplexor also controlled by the ICC, which selects from one of the buffers or the bypass path.

- *Header Modification Circuit (HMC)*. This component makes minor changes to the header as specified by the ICC. If the rot bit is asserted, words 1–3 of the routing field are rotated, with word 1 becoming word 3, word 2 becoming word 1 and word 3 becoming word 2. If the cpy bit is asserted the packet is sent to both output ports and the fanout fields of the copies are modified. The bcn bit determines which copy gets the “extra” when the fanout value is odd. The en0 and en1 signals control the passage of data onto the output links, with en0 enabling output 0 and en1 enabling output 1.

Figure 4.7 details the Input Control Circuit. The ICC contains several major components. The Header Register and Decode Logic (HRDEC) latches various fields of an incoming packet’s header and decodes those fields into six bits. The cpy bit is 1 if the packet must be copied to both outputs. The bcn bit specifies which output receives the “extra” when the fanouts of the two copies are modified. The rot bit is 1 if words 1 to 3 should be rotated. The rn bit is 1, if there is an incoming packet. The r0 bit is 1, if output 0 is required and the r1 bit is 1, if output 1 is required.

The buffer control registers BCREG0, BCREG1 store the decoded control bits for packets stored in BSR0 and BSR1. Each BCREG has six data inputs and six tri-state data outputs. In addition, the rn signal has a non tri-state output. The BCREGs have two control inputs. If latch is high at t16, the input control bits are latched. When sbc is high, the six stored bits are placed on the tri-state outputs.

The PLA at right provides overall control of the ICC. At the start of the epoch it selects one of the HRDEC or BCREGs to provide a request vector to the Output Control Circuit. Then, based on the response, it controls the steering of data to and from the buffer shift registers and controls updating the BCREGs. It also generates the upstream grant signals. The latches at the top of the figure simply hold the control signals for the duration of the epoch and are latched at the times indicated.

The fifo to the right of the PLA is used to keep track of the order of packets stored in the buffers. The fifo is two bits wide and two deep. The output of the fifo gives the number of the buffer containing the packet which is to be output first.

4.4. Packet Processor

The Packet Processors (PP) form the interface between the external fiber optic links and the switch module’s internal data paths. They perform all the link level

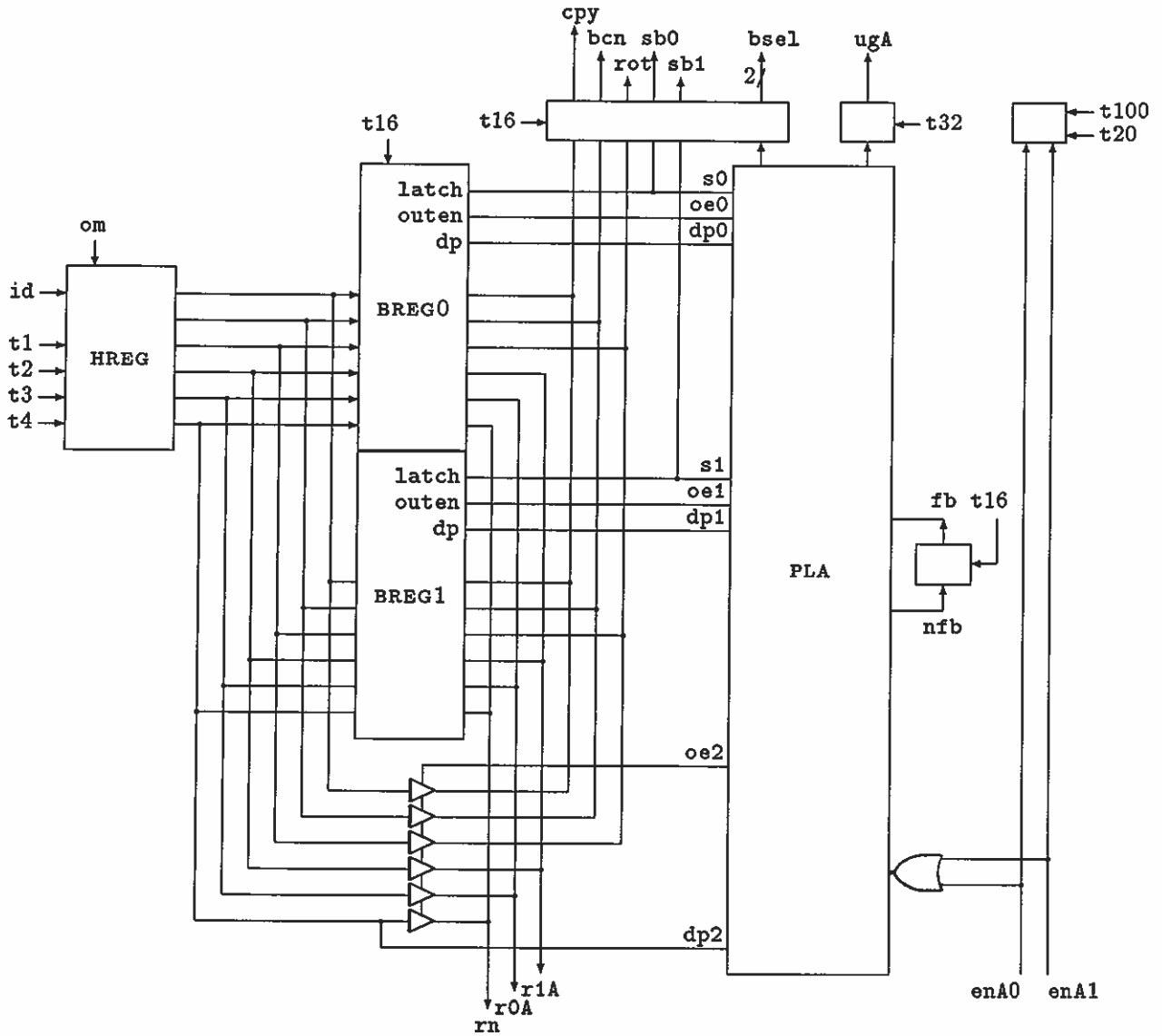


Figure 4.7: Input Control Circuit

protocol functions, including the determination of how packets are routed.

External Interface

The external leads of the packet processor are shown in Figure 4.8 and summarized briefly below.

- *Upstream data from SF* (ud). Data from switch fabric. Nine bits wide including parity.

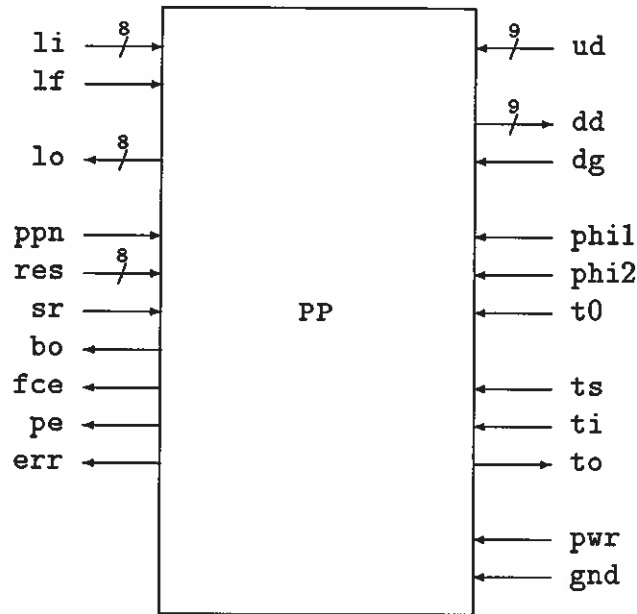


Figure 4.8: External Interface for Packet Processor

- *Downstream data to SF (dd)*. Data to switch fabric. Nine bits wide including parity.
- *Downstream grant from SF (dg)*. When asserted, allows PP to transmit packet in subsequent epoch.
- *Data from link (li)*. Data stream from FOL. Eight bits wide.
- *Link framing (lf)*. Link framing signal. Goes high at start of packet.
- *Data to link (lo)*. Data stream to FOL. Eight bits wide.
- *PP number (ppn)*. Eight bit number identifying PP.
- *Reset (res)*. Resets the entire PP when it is asserted, causing any packets stored in the PP to be discarded.
- *Soft reset (sr)*. Resets PP error flags.
- *Buffer overflow (bo)*. Asserted whenever a packet is discarded by the PP due to buffer overflow.

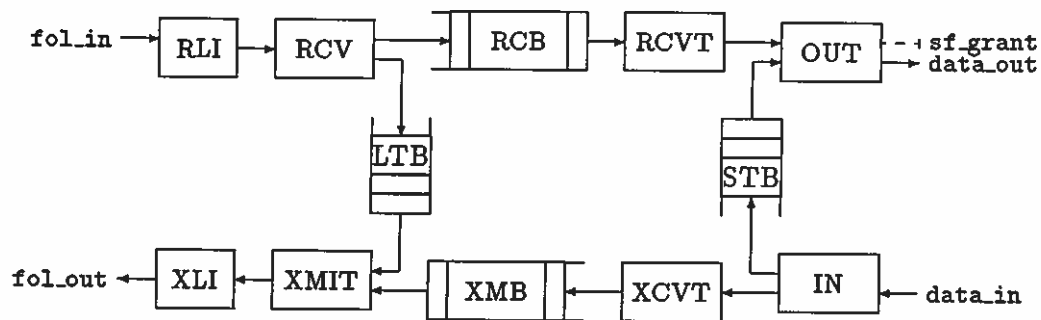


Figure 4.9: Packet Processor Circuit

- *FC error (fce)*. Asserted when the PP receives a packet containing a bad frame check field.
- *Parity error (pe)*. This signal is asserted whenever the PP detects a parity error.
- *Error (err)*. Asserted when the PP detects any error, including those signaled above.
- *Clock (phi1,phi2)*. Two-phase, non-overlapping clock.
- *Start of epoch (t0)*. Goes high when first word of packet is present on udleads.
- *Test shift (ts)*. Shift lead for controlling shifting of test data.
- *Test in (ti)*. Input lead for test data.
- *Test out (to)*. Output lead for test data.
- *Power (pwr)*.
- *Ground (gnd)*.

Global Operation

The processing of packets by the PPs is determined by the PTYP field for external packets (received from FOL) and by the OP field for internal packets (received from SF).

- *External Data Packet.* Converted to internal format, with the routing field determined by a lookup in an internal *Logical Channel Translation Table* (LCXT). The packet is then transmitted to the switch fabric.
- *External Link Test Packet.* The PTYP field is changed to external data packet, and the packet is returned on the outgoing FOL.
- *External Control Packet.* Converted to internal format, with the LN field set to 0 and the RC set to ordinary data packet. Transmitted to SF.
- *Internal Data Packet.* Converted to external format, with contents of internal LCN field transferred to external LCN field. Transmitted to FOL.
- *Switch Test Packet.* The RC field is set to 0 and then the first five words of the packet are moved to the end of the packet and everything else shifted up. In other words, the whole packet is rotated by five words. The packet is then returned to the SF.

Internal Components

A block diagram of the PP appears in Figure 4.9. The various components are described briefly below.

- *Buffers.* The PP contains four packet buffers. The *Receive Buffer* (RCB) is used for packets arriving from the FOL and waiting to pass through the SF. The *Transmit Buffer* (XMB) is used for packets arriving from the SF that are to be sent out on the FOL. The *Link Test Buffer* (LTB) and *Switch Test Buffer* (STB) provide paths for test packets used to verify the operation of the FOL and SF respectively. The RCB has a capacity of 16 packets, the XMB has a capacity of 32. The LTB and STB can each hold two packets. Together, the four buffers require a total of about 35 Kbits of memory.
- *Receive Link Interface* (RLI). Converts the incoming optical signal to an eight bit electrical format, synchronized to the local clock.
- The *Receive Circuit* (RCV). Checks incoming packets for errors, adds parity, strips off CRC, routes test packets to the LTB and other packets to the RCB.
- *Receive Conversion Circuit* (RCVT). Adds five bytes of header information to the front of each packet received from the RCB.

- *Output Circuit (OUT)*. Performs logical channel translation, using an internal *Logical Channel Translation Table (LCXT)* and sends packets to the SF. Also reads test packets, LCXT read/write packets and PP parameter block read/write packets from the STB and processes them appropriately.
- *Input Circuit (IN)*. Routes internal data packets to the XCVT and others to the LTB. Performs the rotation required for switch test packets.
- *Transmit Conversion Circuit (XCVT)*. Removes the first five words of the packet then transfers it to the XMB.
- *Transmit Circuit (XMIT)*. Takes packets from the XMB, adds the SYNC field, strips parity and computes the frame check. Also processes test packets from the LTB.
- *Transmit Link Interface (XLI)*. Converts from eight bit electrical format to optical format.

The RLI and XLI will be implemented separately from the integrated circuit that implements most of the PP functions. These will be implemented using commercially available components. We are currently evaluating the TAXI chip set for this purpose. The circuits that make up the bulk of the PP chip can be divided into three basic types; synchronous streams processors (SSP), packet buffers and lookup tables.

An SSP is a circuit that has several typed I/O ports over which it sends and receives data in a highly synchronous fashion and which transforms the contents of certain fields as the data passes through. These can be readily described in a high level specification language that we have designed and can then be compiled into a special-purpose circuit implementing the given specification. Some details of the language and the translation process are given in the following chapter of this report. The RCV, XMIT, RCVT, XCVT, OUT and IN circuits are all examples of SSPs and our plan is to implement all of them using the automatic translation process just described.

A packet buffer is a memory designed to hold packets. The XMB, RCB, LTB and STB are all specific instances. As with the SSPs, we are designing a program that will automatically generate a specific packet buffer given a description of its desired characteristics (packet length, data path width, timing requirements, etc.). This program will be used to design the particular packet buffers required for the PP. This is described further in the following chapter.

A lookup table is also a memory, but has a somewhat different interface than a packet buffer, since it must offer random access to the table entries rather than

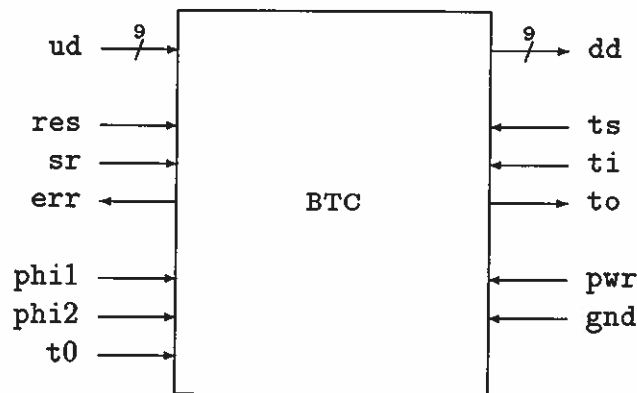


Figure 4.10: External Interface for BTC

implementing a buffer. We will soon begin work on a similar program that generates a lookup table from a given specification. The LCXT is the only lookup table in the PP, but other chips in the system also contain lookup tables, so we felt it advantageous to design a general tool to create them.

4.5. Broadcast Translation Circuit

The Broadcast Translation Chip (BTC) provides unique addresses for each of the copies of a broadcast packet replicated by the copy network. It also provides a hardware assist for updating the table of new addresses for a single broadcast channel.

External Interface

The external leads of the BTC are shown in Figure 4.10 and described briefly below.

- *Upstream data leads (ud)* Incoming data from upstream neighbors. Nine bits wide including parity.
- *Downstream data leads (dd)* Outgoing data to downstream neighbors. Nine bits wide including parity.
- *Reset (res)*. Resets the entire BTC when it is asserted, causing any packets stored in the BTC to be discarded.

- *Soft reset (sr)*. Resets PP error flags.
- *Error (err)*. Asserted when the BTC detects any error.
- *Clock (phi1,phi2)*. Two-phase, non-overlapping clock.
- *Start of epoch (t0)* Goes high when first word of packet is present on udleads.
- *Test shift (ts)*. Shift lead for controlling shifting of test data.
- *Test in (ti)*. Input lead for test data.
- *Test out (to)*. Output lead for test data.
- *Power (pwr)*.
- *Ground (gnd)*.

Global Operation

The BTC's operation depends upon the type of packet passing through it.

- *Ordinary Data Packet*. These packets are passed straight through the main shift register unchanged.
- *Broadcast Data Packet*. The routing field is replaced with a new field selected from an internal *Broadcast Translation Table (BTT)*. The new field is selected using the BCN of the packet.
- *Read/Write BTT Block*. These two packet types are used for updating the BTT in large chunks and for reading it for auditing and testing purposes.
- *Read/Write BCIT Block*. These packets read and update the entire broadcast copy index table.
- *Single Entry Update*. This packet supplies a set of new routing fields for this BCN. The BTC chooses which one to write into the BTT depending on the value of the *broadcast copy index*, which is determined from an internal *Broadcast Copy Index Table (BCIT)*. I[0] specifies the new fanout for the connection, I[1] specifies which group of routing fields are contained in the packet. I[2]–I[65] contains the routing fields.
- All other packets are passed through like ordinary data packets.

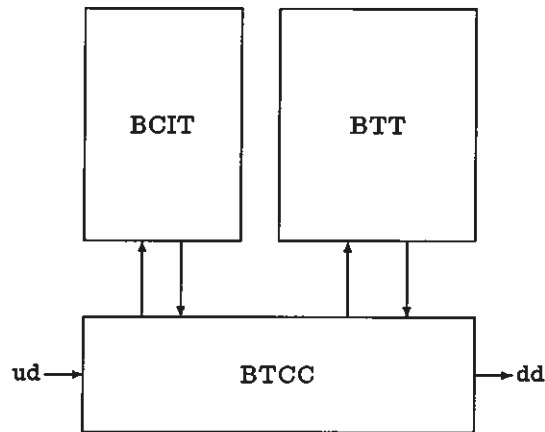


Figure 4.11: Block Diagram of Broadcast Translation Chip

Internal Components

The internal components that make up the BTC are similar to those in the PP. In particular, the BTC can be described as a single SSP along with two lookup tables, one for the BTT and the other for the BCIT. This structure is illustrated in Figure 4.11 and the components are described briefly below.

- **Broadcast Translation Table (BTT).** This is the table used to store the routing information for packets belonging to multipoint connections. The prototype version will consist of 256 entries, each four bytes long, plus parity.
- **Broadcast Copy Index Table (BCIT).** This table is used to compute the broadcast copy index of the BTC, for use in updating the BTT when the fanout of a connection changes. It contains 512 single byte entries, plus parity.
- **BTC Control (BTCC).** This is the control circuit that processes all received packets using the BTT and BCIT.

Our plan is to implement the two lookup tables using a lookup table generator, which is described briefly in the next section. The BTCC will be implemented as a synchronous streams processor using the program we are developing for that purpose.

5. Tools for Design of Communication Circuits

Faculty
Research Associate
Graduate Student

Jonathan Turner
Pierre Costa
Neil Barrett
George Robbert
James Sterbenz
Einir Valdimarssen

The implementation of the prototype packet switch for the ACS project will require several custom VLSI chips. In the past year, we have designed preliminary versions of two chips and gained considerable insight into both the impact of low-level design issues on architecture and on the design process itself. In this chapter, we review our efforts at constructing special purpose tools to reduce the amount of manual effort associated with the design of integrated circuits. Our most ambitious effort in this area is the design of a circuit generator for a particular class of circuits that arise frequently in our work and which we refer to as *synchronous streams processors*. We have also begun several other projects which seek to automate or partially automate other design tasks.

5.1. Synchronous Streams Processors

Many of the circuits required in a fast packet switching system contain a large number of functional modules that accept packets on one or more input ports, modify the packet headers and transfer the packets to one or more output ports. The various modules operate in tight synchronism because of the use of fixed length packets. We have come to view each of the specific modules as special cases of a generic *synchronous streams processor* or SSP.

An SSP, is a module with one or more typed input and output *ports*, a local clock synchronized by external timing signals and a function which can be described in

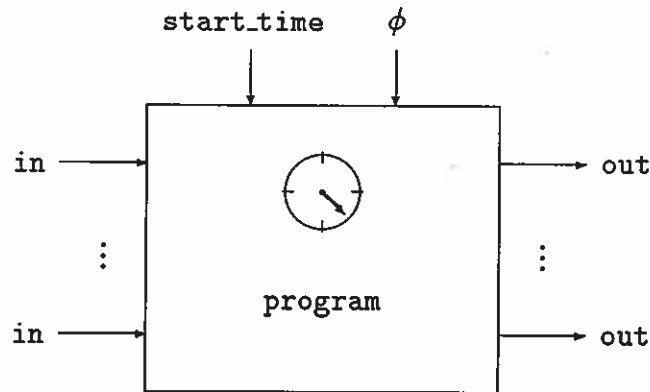


Figure 5.1: Generic Synchronous Stream Processor

a style similar to a conventional programming language (see Figure 5.1). The local clock is set to 0 when the external synchronization signal *start_time* is received, and is then incremented on every tick of the global system clock ϕ . The period between successive *start_time* signals is referred to as an *epoch* and all events happen at specific times during an epoch.

Each port has a type associated with it. The base type is *bit* and complex types can be constructed using arrays and structures. As an example, the following declaration defines the format of an internal packet in the BPN prototype.

```

struct ipfmt {
    bit op[5], rc[3];
    bit fan_ln[8];
    bit bcn_ilcn[16];
    bit src[8];
    bit fill[4], ptyp[4];
    bit elcn[16];
    bit info[72][8];
};

```

In addition to its type, a port has a *start time* and a *width*. The start time defines at what point in each epoch the data item defined for that port begins to appear on the port. The width of the port defines the number of bits available to carry the data. These pieces of information are sufficient to define when in an epoch and where on a port, specific items of data appear. This allows a designer

to describe the function of an SSP in terms of actions on port fields, ignoring the details of timing and bit location.

We now turn to a simple example to illustrate how an SSP can be described. The circuit we will describe combines the functions of the IN and XCVT circuits of the packet processor. It accepts an input packet from the switch fabric on one port and based on the control field directs the packet to either the XMB or STB. For test packets, the RC field is set to 0 and the packet is rotated by five words. In addition, packets sent to the XMB are placed in a format that is intermediate between the internal and external formats; basically, it is the internal format with the first five words removed. As part of this transformation, the logical channel number in the ILCN field is copied to the ELCN field.

```

struct hpfmt {                // Hybrid packet format
    bit fill[4], ptyp[4];
    bit elcn[16];
    bit info[72][8];
};
module inxcvt(port[8] struct ipfmt <sfp@0, >ltbp@2,
              port[8] struct hpfmt >xmbp@6 )
    if sfp.op == OP_DATA ->
        xmbp.fill = sfp.fill;
        xmbp.ptyp = sfp.ptyp;
        xmbp.elcn = sfp.bcn_ilcn;
        xmbp.info = sfp.info;
    | sfp.op == OP_STEST ->
        ltbp:(0,74*8) = sfp:(5*8,79*8);    // Rotate
        ltbp:(75*8,79*8) = sfp:(0,4*8);
        ltbp:(75*8,75*8+2) = 0;           // Clear rotated RC
    | sfp.op != OP_DATA && sfp.op != OP_STEST ->
        ltbp = sfp;
    fi;
end

```

The example module defines three eight bit wide ports; sfp is an input port (indicated by <) carrying data in mpfmt, starting at time 0; ltbp is an output port, also carrying data in mpfmt, starting at time 2; xmbp is an output port carrying data in hpfmt, starting at time 6. The program specifies the appropriate action based on the OP field of the incoming packet. The assignments define the contents of the various output fields. Unspecified output fields are filled with zeros. The notation port:(i, j) refers to a range of bits on the given port, providing a simple low level mechanism for rearranging large blocks of a packet. Notice that the only

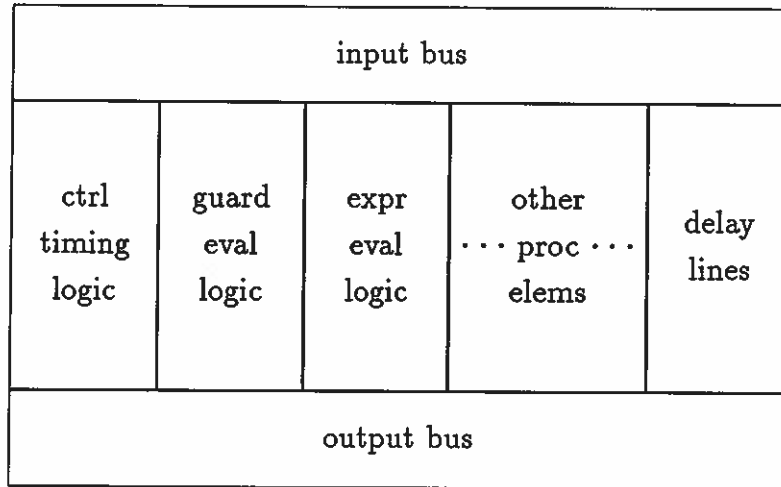


Figure 5.2: Target SSP Architecture

times that must be defined explicitly are the times at which data items start to flow across ports.

The simple paradigm of typed, synchronous ports can also be used to define control signals that must be exchanged between different modules. This allows us to define more complicated interfaces such as are required on the output side of various buffers. Modules can also have local variables that can be used to save information across time epochs.

5.2. Implementation of SSPs

SSPs that perform simple functions, as are typical in the Packet Processors, fit nicely into a common architecture illustrated in Figure 5.2. This architecture supports several input and output ports of varying widths. Input ports connect to a common input bus and outputs to a common output bus. Between these are a set of processing elements (PE). Each processing element has data registers which latch selected input fields. The *guard evaluation logic* in addition, contains the combinational logic to evaluate the conditions in if-statements. The *expression evaluation logic* evaluates expressions on the right side of assignments. The *delay lines* are used to delay the passage of certain fields to the output bus in order to satisfy timing constraints. The control and timing element provides timing signals for latching input data and controlling access to the output bus.

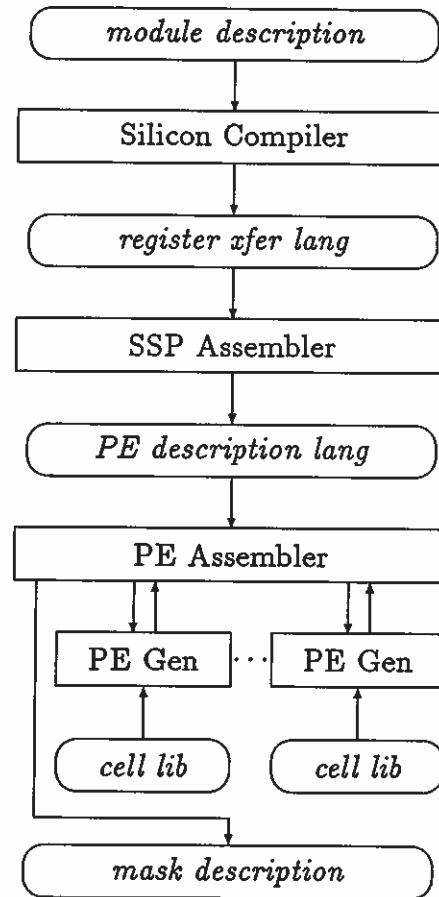


Figure 5.3: Structure of A Silicon Compiler

Our objective is to develop a circuit generator that takes a high level description of an SSP and creates a circuit implementing it, by tailoring the target architecture. We have divided the translation into several parts as illustrated in Figure 5.3. The *compiler* takes the high level module description and translates it to a simple *register transfer language*. This is further processed by an *SSP assembler* which translates it further to a PE description language. This is further processed by a *PE assembler* which generates the actual mask-level description of the module, using a library of standard cells and a set of PE generators, which include existing tools such as a PLA generator. A great deal of progress has been made on both the compiler and the assembler in the last six months. An early version of the compiler can now generate many simple circuits and work is continuing to make it more general and robust.

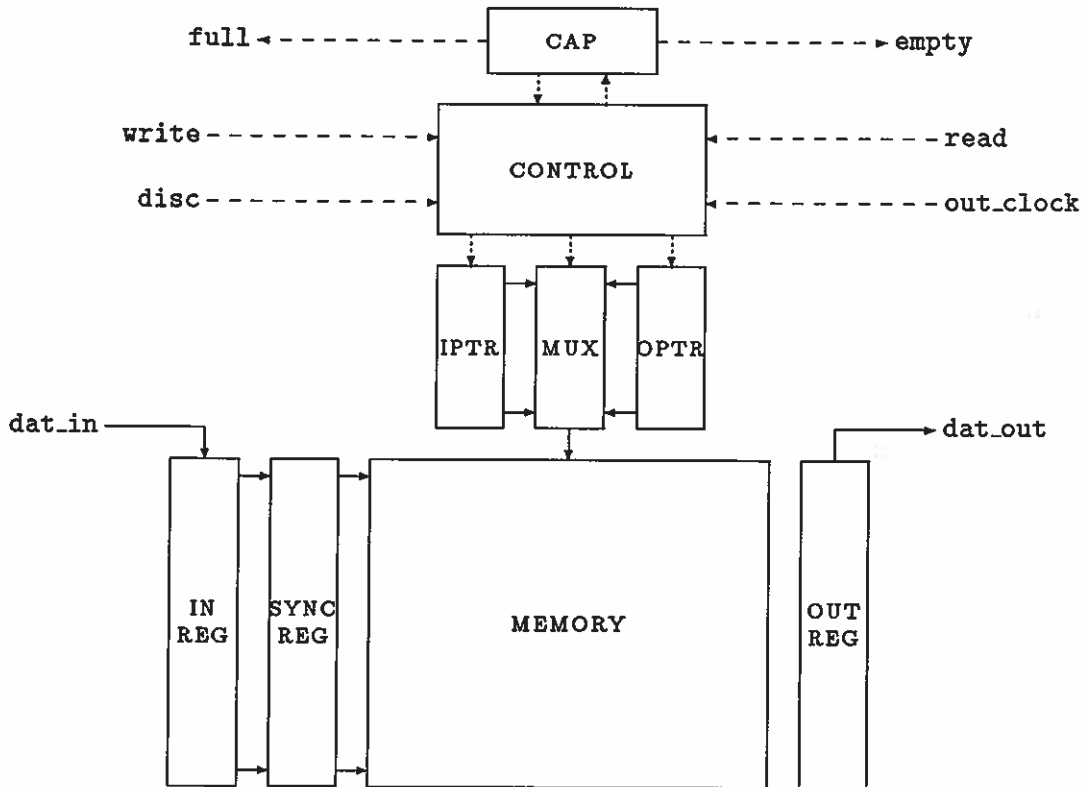


Figure 5.4: Packet Buffer

5.3. Tools for Constructing Memories

Two general memory types arise often in the circuits for the broadcast packet switch; packet buffers and lookup tables. A packet buffer is a first-in-first-out buffer for storing packets. During a given packet cycle or epoch, one packet can be written to the buffer and one read out. The overall structure of a typical packet buffer is shown in Figure 5.4. Its external interface includes input and output data ports and several control and status signals. The memory array is split into multiple bit planes and uses static memory; dynamic memory was avoided to reduce control complexity and because simulation studies indicated that the required refresh intervals might prove difficult to achieve. Reading and writing to the memory array take place at specific times during a packet cycle or epoch. Incoming packets enter input shift registers and when the packet has been shifted in, the content of the input shift registers is transferred to buffer registers and from there to the appropriate place in memory. Outgoing packets are read from memory

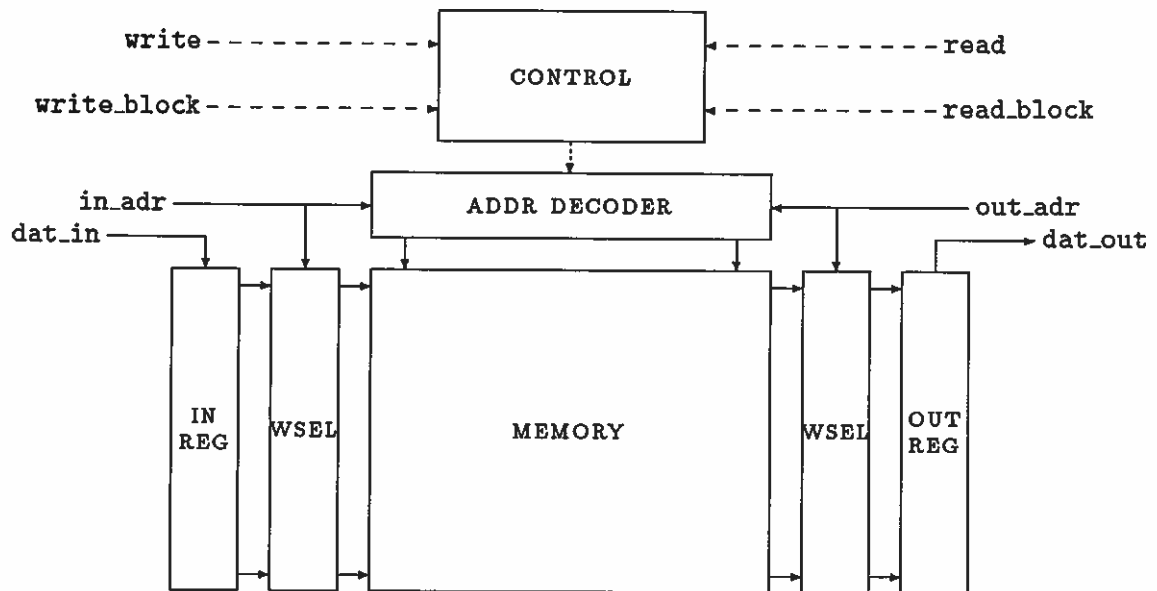


Figure 5.5: Lookup Table

to an output shift register and then shifted out over the output port. The control section includes memory read and write pointers and a counter that keeps track of the number of packets in the buffer. We are writing a program that will take as input a specification of a particular packet buffer. Parameters that the user can specify include the size of the packets, the number of packets stored in the buffer, the data path widths and the target aspect ratio.

A lookup table is a second memory type, used for storing a table of data. The entries in a lookup table can be accessed randomly, using an externally supplied address. As with the packet buffer, they can be read or written just once per packet cycle. It is also possible to access them in large blocks rather than single entries. A block diagram of a lookup table appears in Figure 5.5. Incoming data enters on an input data port at a specific time in an epoch and is transferred to memory at a location specified by the address signals. Similarly, data is read from a specified memory location at a specific time in an epoch and then transferred through the output data port. The program for the lookup table will take as parameters the number and size of the table entries, the size of blocks for the large block transfers, the data path widths and the target aspect ratio.

5.4. Other Tools

Several other tools for assisting in circuit development are being planned. One is a program for generating registers and counters. User-specified options will include number of bits, up-counter, down-counter, shift capability, inverted outputs and tri-state outputs, among other things. As a first step in this work, we have developed a library of standard cells that can be easily assembled into a variety of counter types.

Another tool that would be useful is a program that generates circuits implementing small amounts of random logic. Often in a complex circuit, there is a need for a few gates worth of logic to generate control signals. The amount of logic is too small to warrant a PLA and consequently the required gates are typically hand-crafted for the particular circuit. This is reasonable, but requires more time and effort on the designer's part than is really warranted. A better approach would be to have the circuit generated by a program. We plan to examine the feasibility of such a circuit generator in the coming year.

One of the more time consuming parts of an integrated circuit design is the simulation performed to help verify the correct operation of the circuit. Much of this work is tedious, repetitive and error-prone. We are considering strategies for reducing the time and effort consumed in this phase of the process. One promising idea is to develop a high level circuit simulator that would allow the user to specify the operation of the circuit and a set of test inputs in a high level notation similar to a programming language. The program would determine the appropriate outputs for the user's test inputs and would then generate the required low level test vectors required by the various circuit simulation tools. The user could then run the circuit simulator and compare its output to the predicted output. Such a tool could be used in conjunction with the circuit generator for the synchronous streams processor, to verify that the generated circuits are correct and to facilitate timing simulations for performance evaluation.

6. Connection Management

Faculty
Graduate Students

Jonathan Turner
Victor Griswold
Mark Hunter
David Wexelblat

Connection management refers to the collection of algorithms, data structures and protocols used to create and maintain connections among users. In conventional networks, connections join two endpoints. In multipoint networks, connections may join an arbitrary number of endpoints. Several types of connections appear to be useful, including point-to-point connections and simple broadcast connections having one transmitter and many receivers. Connections in which all participants can both transmit and receive are also useful for conferencing and LAN interconnect applications among others.

As one considers applications of multipoint communication, one soon realizes that what is needed is a general multipoint connection capability that realizes point-to-point, broadcast and conference connections as special cases. In section 6.1 below, we describe a flexible scheme for specifying multipoint connections that provides the network with the information needed to allocate resources appropriately. In section 6.2 we briefly describe both an access and an internal protocol for manipulating such connections and in section 6.3 we report on a simulator that has been developed to test these protocols and to provide a foundation for future developments.

6.1. Specification of Multipoint Connections

As indicated above, there are a variety of useful multipoint connection types. Rather than have the network implement each type individually, we have chosen to define a general capability that includes each specific type as a special case.

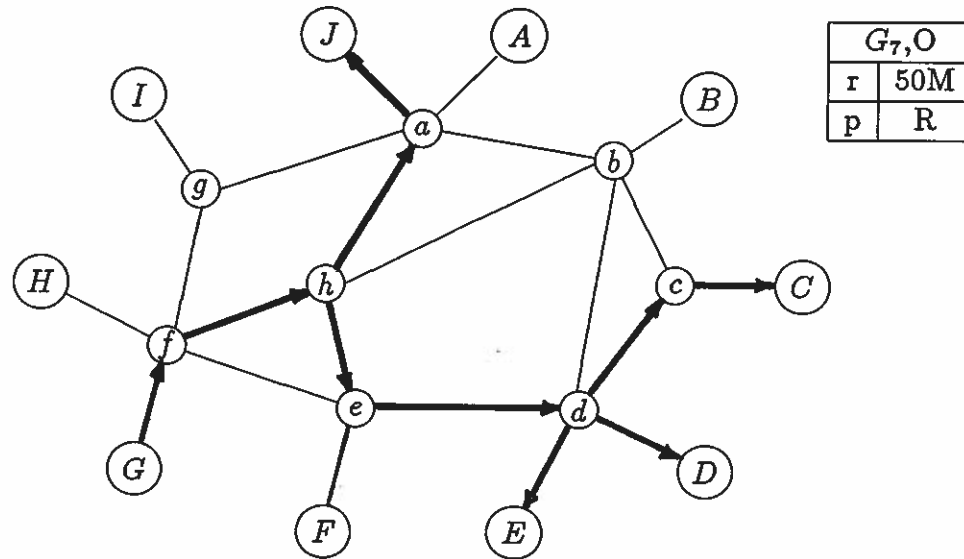


Figure 6.1: One-to-Many Connection

We introduce this method of specification by giving several examples. We start by describing a simple one-way broadcast connection, illustrated in Figure 6.1. The connection has a single transmitter G and receivers C , D , F and J . The internal nodes in the diagram represent switching systems. At various internal nodes, the stream of packets originating at G is replicated and forwarded to the appropriate destinations. The connection induces a tree in the network, in much the same way that a point-to-point connection induces a path in a conventional network. The table in the upper right-hand corner of the diagram summarizes some global information describing this connection. The G_7 at the top is the *connection identifier*, which is a globally unique name identifying this connection and distinguishing it from all other connections in the network; the motivation for having a connection identifier will be explained below. One simple way of providing such an identifier is to use the address of the *owner* of the connection together with an integer distinguishing this connection from others that the owner may also be participating in. The owner of the connection is just that termination that is responsible for the connection and controls access to it. This scheme has been used in the example, implying that G is the owner, as well as the only transmitter in the connection. The 50M denotes a *rate specification* of 50 megabits per second. In a real network, a more complex rate specification is required, allowing specification of peak rate, average rate and some measure of “burstiness,” but we will ignore this issue here. Each endpoint participating in the connection can have *transmit-only*

permission, *receive-only* permission or *transmit/receive* permission. The permission concept provides the basic mechanism needed to allow the specification of general multipoint connections, that can be tailored to different applications. The network uses the permission information to allocate resources, (primarily link bandwidth) appropriately. The *R* at the bottom of the table defines the *default permission* to be receive-only, meaning that whenever an endpoint is added to the connection it is initially assigned receive-only permission; this can of course be changed by the owner if some other permission is required.

The example illustrates a connection that might be appropriate for distributing an entertainment video signal. To establish such a connection, *G* would send a control message to the network, describing the type of connection required. At that point it could begin transmitting on the connection, but initially there would be no one to receive the signal. Endpoints can be added to the connection in one of two ways. First *G*, as the owner, can send a message to the network asking that a particular endpoint be added. In response, the network would send a *connection invitation* to the specified endpoint and if the endpoint agrees (by exchange of control messages) to join the connection, the network would allocate the necessary resources to include the new endpoint in the connection. For entertainment video signals, a more appropriate way of adding an endpoint is at the endpoint's request. That is, an endpoint could send a control message to the network, requesting that it be added to a specified connection. To make such a request, the endpoint must specify the appropriate connection identifier. For entertainment video signals, this information would typically be widely available and could be built into terminal equipment, or programmed in, as appropriate. In response to such a request, the network would first search for the nearest place that the specified connection is available and then attempt to add the new endpoint by creating a branch at that point.

Addition of new endpoints from "outside" the connection raises the need for some form of authorization. In the example, the *O* at the top of the table specifies that this connection is *open*, meaning that anyone who wishes to join the connection may do so without explicit authorization from the owner. This would probably be the appropriate specification for a commercial video broadcast. Other options include *closed*, meaning that no one can join from outside and *verify*, meaning that outsiders may join, but only after getting explicit permission from the owner.

This example has illustrated the essential notions of the multipoint connection mechanism. A point-to-point connection for voice communication can be readily described using these mechanisms. In that case, the rate specification might be 50 kilobits per second rather than 50 megabits, the default permission would be *TR*, for transmit/receive and the outside access specification would be *C* (closed). Notice that in neither example is there any need for the switching systems supporting

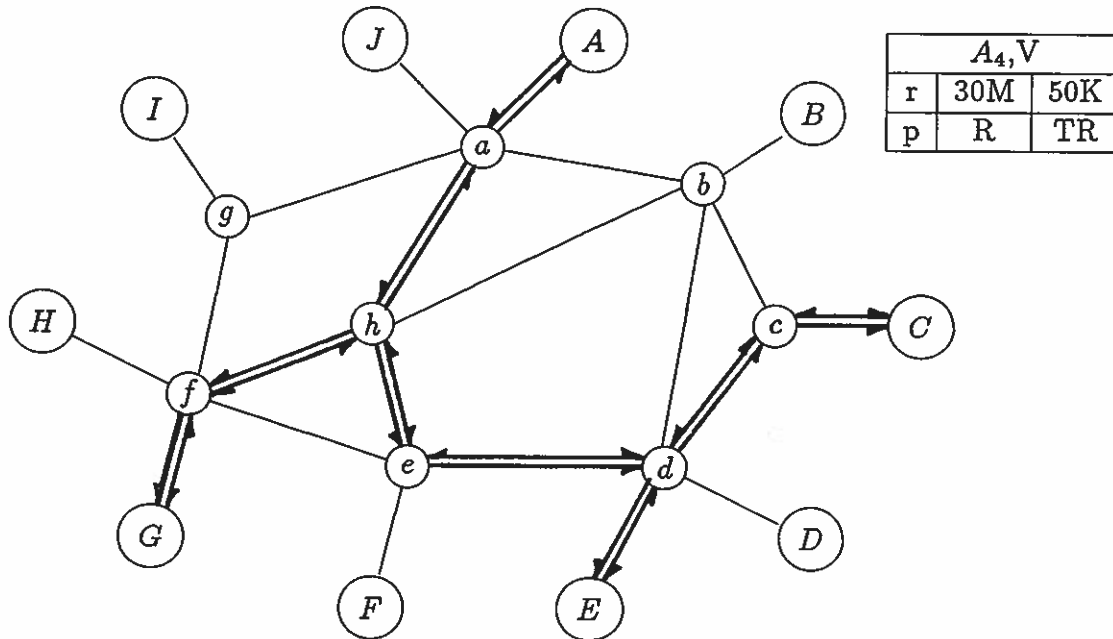


Figure 6.2: Connection for Video Lecture

the connection to have detailed knowledge of the application, such as whether the signals carried are voice or video. Such information is required in the terminations and possibly network interface equipment (depending on the desired form of access), but is not needed anywhere in the core of the network. This *application-ignorance* is important for maintaining the flexible nature of the network. By keeping that information from the network core, we ensure that changes in the applications cannot affect the network's operation in any fundamental way.

Figure 6.2 gives an example of a more complicated connection requiring two related but separate channels within a single connection. This connection is intended for use in a video lecture situation. It includes a downstream channel from *A* to *C*, *E* and *G*, which has a rate of 30 megabits per second and a default permission of receive-only. It also has a second channel, which can be used for upstream audio, allowing the audience members to direct questions to the speaker. The audio channel has its own rate and default permissions specified. The network allocates link bandwidth separately for the two channels, but routes them through the same set of switches, for convenience in handling addition of new endpoints. The concept of multiple channels within a connection is an important extension of the basic multipoint connection mechanism that can be useful in more complex applications.

6.2. Multipoint Control Protocols

This section briefly describes the access and internal network protocols for establishing and modifying multipoint connections. The access protocol is used for control communication between the network and the terminations. At this level of abstraction, the network is viewed as a monolithic entity. The internal network protocol is used for control communication among the internal network nodes. This protocol must ensure that concurrent control operations taking place in a given connection are handled in a consistent fashion.

Before we describe the protocols, we must mention the basic mechanisms for providing reliable communication. The reliable transmission of control messages is handled as a low level function separate from the higher level control protocols. That is, there is a low level protocol that detects lost control messages and retransmits them as necessary, without explicit involvement of the higher level protocols. We omit any detailed discussion of this low level protocol.

Access Protocol

The access protocol supports four general classes of operations on a multipoint connection: commands, queries, offers and notifications. A *command* is a directive from the user to the network that requests establishment or modification of a connection. A command consists of two messages, a *request* from user to network and an *acknowledgement* from network to user. The major command types are listed below.

- `open_con`. Requests opening of a new connection. Parameters include the external accessibility, the rate and default permissions for the initial channel, and the owner permissions. The acknowledgement includes the logical channel number for the initial channel. This channel is also used for all subsequent control messages relating to this connection.
- `close_con`. Requests that all endpoints be removed from a connection and all resources associated with the connection released.
- `open_chan`. Requests a new channel in an existing connection. Parameters include the rate and default permissions and the owner permissions. The acknowledgement includes the logical channel number.
- `close_chan`. Requests that all resources associated with a given channel be released.

- `add_EP`. Requests addition of a new endpoint to a connection. Parameters include the identity of the new endpoint's termination, permissions for each channel and an initial control message to be passed transparently to the new endpoint.
- `join`. Requests that the termination be added to an existing connection. Parameters include the identifier of the desired connection and the expected rates and default permissions for each of the connection's channels. The acknowledgement includes the actual rates and permissions for the channels and the logical channel numbers to be used for each channel.
- `drop_EP`. Requests removal of an endpoint from the connection. Parameters include the identity of the endpoint to be dropped.
- `drop_out`. Requests removal of the requesting endpoint. No parameters.
- `change_con`. Requests modification of global connection parameters such as the external access specification.
- `change_chan`. Requests modification of the rate or default permissions associated with a particular channel.
- `change_EP`. Requests modification of the permissions associated with a particular endpoint.

Most commands can be issued only by the owner of the connection; exceptions are the `join` and `drop_out` commands.

A *query* is a request for information from either the user to the network or from the network to the user. A query can relate to a connection the user is currently participating in or to an unrelated connection. A query consists of two messages, the request and the reply which contains the desired information. Queries from users are strictly informational and have no effect on the state of any connections. The only query currently defined is the `check_join`, described below.

- `check_join`. The request is sent from the network to the owner of the connection informing it that some endpoint has requested entry to the connection and seeking verification. The response includes acceptance or refusal and, in the case of acceptance, the channel permissions for the new endpoint.

An *offer* is made by the network to a user, and affords the user an opportunity to do something but does not oblige the user in any way. An offer consists of three messages, the initial request from the network to the user, an acknowledgement that either accepts or rejects the offer and a *confirmation* from the network to the

user. The confirmation is used only when a user accepts an offer and informs the user that the actions necessary to carry out the offer have been completed. The network may also send the user an *abort* message, cancelling the offer at any time before the confirmation. The offers currently defined are listed below.

- *invite*. The request message invites the user to join a connection. It includes all the connection parameters, plus the prospective endpoint's permissions and an initial message from the originating termination. If the user accepts, the network replies later with a confirmation.

A *notification* is a message from the network to a user of some change that has taken place in a connection. A notification consists of just the one message from the network. No response is expected or permitted. The notifications currently defined are listed below.

- *announce_EP*. Sent from network to owner when a new endpoint has joined a connection and the owner has requested notification of all endpoint changes.
- *announce_drop*. Sent from network to owner when an endpoint drops out of a connection and the owner has requested notification of all endpoint changes.
- *announce_change*. Sent from network to endpoint to inform it of a change in the connection's channels or the endpoint's permissions.
- *dropped_from_con*. Sent from network to endpoint to inform it that it has been removed from an existing connection.
- *con_failure*. Sent from network to owner informing it of failure of the connection. The connection is closed and all resources released. Typically caused by an unrecoverable hardware failure.

Internal Protocol

The network's internal protocol must provide the mechanisms needed to support the operations the user can specify in the access protocol. Because the network is actually a collection of geographically distributed switching systems and not a monolithic entity, some care must be taken to maintain consistency of the network's internal data structures. Multipoint connections raise the possibility of concurrent control operations taking place in a given connection. To ensure that such operations do not lead to inconsistency, we require a general mechanism that makes such concurrent operations occur in an *apparently serial* fashion.

The internal network protocol implements two general classes of operations; queries and transactions. A *query* consists of two messages; the initial request and the reply. A query has no effect on the internal state of the target node, although the reply may well affect the state of the originating node. Query messages may be forwarded from one node to another; if the node that first received the message cannot reply it relays the message to another node that may be able to reply and then forgets about the message. The node that ultimately replies to the query must be able to do so using only local information and replies directly to the originating node.

A *transaction* consists of a sequence of messages used to effect some change in a connection that affects two or more nodes. Transactions use four types of messages; requests, acknowledgements, commits and aborts. A request is sent from one node to another to request its assistance in carrying out some operation. The receiving node sends an acknowledgement when it is prepared to complete the requested action or when it determines that it cannot carry out the request. A *commit* message is sent by a node in order to finalize changes requested and acknowledged earlier, and is the mechanism that ensures that all nodes involved in the transaction remain consistent. A node can drop out of a transaction at the time it sends its acknowledgement. In such a case, no commit message will be sent to the node. This option is most commonly used when the node cannot complete the requested action, but can also apply to other situations where the node is unaffected by the eventual outcome of the transaction. Typically, a transaction is initiated in response to a user command. The internal network node adjacent to the user's termination sends request messages to one or more other nodes in the connection, which in turn may send requests to others. When the first node receives all outstanding acknowledgements and determines that it can proceed with the requested operation, it sends commit messages to those nodes that it sent requests to and that are still in the transaction. These in turn send commits to the nodes that they sent requests to, and so forth. It is during this *commit phase* of the transaction that resources identified and reserved during the *request phase* are actually allocated to the connection.

A transaction or part of a transaction may be aborted if the resources required to carry out the operation are not available. Aborts can also take place as a result of one transaction interrupting another. Because of the possibility for concurrent transactions, we require a mechanism by which one transaction may preempt another in an orderly way. If a transaction request is received at a node at which another transaction (for the same connection) is in progress, one of two things may happen. If the new transaction has a lower priority than the current one, the request message is simply queued until the current transaction completes. If the new transaction has a higher priority, the current transaction must be interrupted. To ensure serializability, this requires rolling back the current transaction and restor-

ing the connection state at that node to the state that existed prior to receipt of the first transaction request. This rollback involves release of any resources that have been reserved by the transaction and transmission of abort messages to other nodes that have been sent request messages and which are still in the transaction. After the rollback, the request message for the first transaction is queued to allow it be restarted later and the request message for the new transaction is processed. Preemption may only occur up until the time that a node sends an acknowledgement to the node that first sent it the request. New transaction requests received after this point are queued until the current transaction completes, typically upon receipt of a commit. The transactions currently defined are described briefly below.

- `net_add_EP`. This transaction implements an `add_EP` command. The request message includes the connection identifier, channel rates and permissions, identity of the target termination and permissions for the target endpoint. Nodes that are not adjacent to the target termination extend the transaction in the direction of the target. The node adjacent to the target sends it an invite message. If the target accepts the connection, positive acknowledgements are sent back along the path followed by the requests. When the acknowledgement is received at the origin of the transaction, it commits the transaction and sends a positive acknowledgement to the termination that sent the original `add_EP` message. If the target rejects the connection, negative acknowledgements are sent back to the origin and no commit phase occurs.
- `net_drop_EP`. This transaction implements a `drop_EP` command. The request message is sent from the node adjacent to the owner to the node adjacent to the target endpoint, which responds with a positive acknowledgement. When the originating node receives the positive acknowledgement, it initiates the commit phase. During the commit phase, resources are released and the target node is sent a `dropped_from_con` message.
- `net_drop_out`. This transaction is initiated by an internal node upon observing that it has only one link in the connection, and that link does not connect to the owner. The request message is sent to its one neighbor. That node acknowledges and then the first node initiates the commit phase at which point resources are released. This transaction effectively prunes connections following removal of endpoints.
- `net_open_chan`. This transaction implements an `open_chan` command. Request messages are sent to all nodes in the connection, following the tree structure of the connection. These requests are sent in parallel to speed processing. The request messages include the rate and default permissions for

the new channel. Each node that receives a message reserves the resources for the new channel on its downstream links and sends requests to its downstream neighbors. If all nodes can allocate the required resources, positive acknowledgements are sent back up the tree and eventually the originating node receives the required positive acknowledgements. It then sends commit messages, which cause the reserved resources to be allocated. The transaction can fail if the required resources for the new channel are unavailable at some node. In the event of failure, an abort is used to release reserved resources.

- `net_close_chan`. This transaction implements a `close_chan` command. It operates similarly to `net_open_chan` but releases resources associated with an existing channel. The actual release of resources occurs during the commit phase.
- `net_close_con`. Implements a `close_con` command. Request messages are sent in parallel to all nodes in the connection, following the tree structure. Resources are released during commit phase.
- `net_join`. This transaction implements a `join` command. The request message parameters include the connection identifier and the expected channel rates and permissions. The transaction is routed through the network until a node in the desired connection is found. If the external access permission is open, positive acknowledgements are then sent back along the path followed by the request messages. These acknowledgements carry the actual connection parameters. When the acknowledgement is received at the originating node, it initiates the commit phase, causing resources reserved earlier in the process to be allocated. If the connection requires verification by the owner, the transaction is extended to the node adjacent to the owner, which seeks the necessary verification with a `check_join` query.
- `net_change_con`. Implements a `change_con` command. Used to modify global connection parameters such as external accessibility. Request messages propagate through connection in parallel following tree structure.
- `net_change_chan`. Implements a `change_chan` command. Used to modify rate or default permissions of a channel. Request messages propagate in parallel following tree structure.
- `net_change_EP`. Implements a change to the permissions of a particular endpoint.

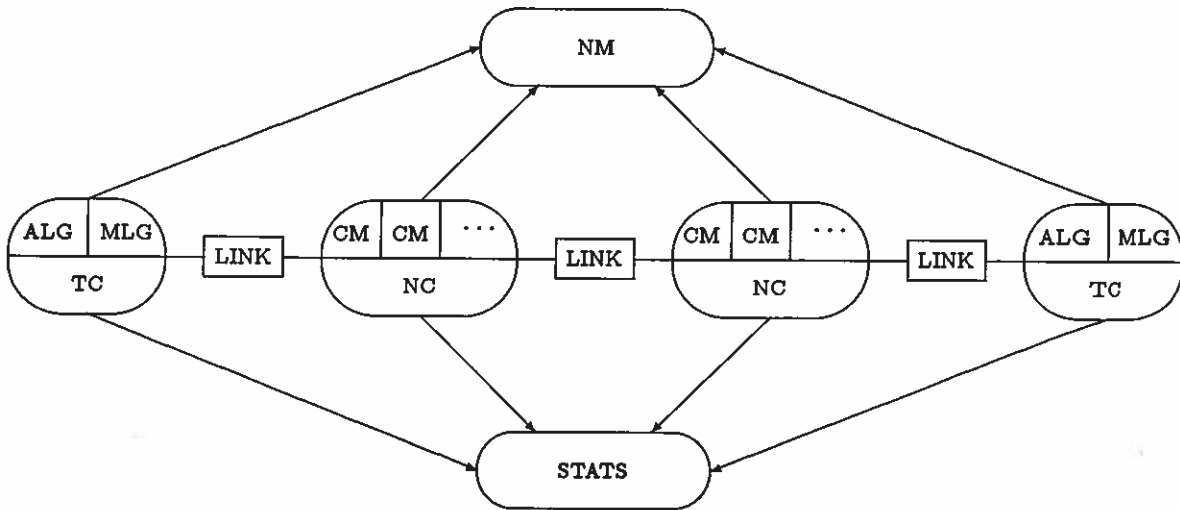


Figure 6.3: Connection Management Architecture

6.3. Prototype Connection Management Software

We have designed and implemented a preliminary connection management software system based on the connection model and protocols described above. This prototype software is implemented as a simulator that allows us to configure arbitrary networks and simulate the interaction of the nodes and terminations. The simulator can be operated in an interactive mode, allowing a user at a termination to send messages of various types to the network and observe the effect of these messages on the state of terminations and internal nodes. It can also be operated in a batch mode, with messages generated by load generators. In this mode, various types of failures can be simulated and the reaction of the connection management software to these failures can be observed.

The program is written in the C++ programming language and uses the C++ task library to simulate concurrent processes. The overall software architecture is shown in Figure 6.3. Each internal node in the network comprises a *node controller* and zero or more *connection managers*. A node has an active connection manager for every connection it is involved in. The connection managers maintain information on the state of the connection, and communicate with each other and the terminations in order to control the connection. The node controller handles the details of resource allocation and communication between nodes. In particular, it ensures that control messages are transmitted reliably. It also provides some support for the transaction protocols described above. In particular, it handles queuing of transaction requests and roll-back of connection managers.

A similar structure exists within the terminations. Each termination comprises a *termination controller* and one or more devices. Devices can simulate a variety of physical devices including telephones, television sets or computers. Control of the simulator is provided through a *network manager*. A user controlling the simulator gives commands to the network manager, which in turn sends messages to a particular device. The network manager also provides the user with access to the data structures within the connection managers and node controllers, allowing him to observe the effect of various actions on the network's internal state.

The simulator also includes mechanisms for simulating link errors and hard failure conditions, thus providing a mechanism for testing the robustness of the protocols. The current implementation, supports a subset of the protocols described above. Subsequent versions will refine the protocols based on what we learn from the current version. We also plan to provide a more convenient user interface, including graphical display of some of the network status information. The software developed for the simulator will form the basis of software for the prototype switch module now being designed.

7. Multipoint Routing

Faculty
Graduate Students

Jonathan Turner
Bernard Waxman

In a packet switched network which uses virtual circuits, the primary goal in routing connections is to make efficient use of the network resources. For example we favor an algorithm which can handle the largest number of connections for a given set of network resources. In a point-to-point network, routing is often treated as a shortest path problem in a graph. Here the network is modelled as a graph $G = (V, E)$ where the nodes of a graph represent switches and the edges represent links. In addition we have two functions $\text{cap}: E \rightarrow \mathbb{R}^+$ and $\text{cost}: E \rightarrow \mathbb{R}^+$ which give us the bandwidth and cost of each edge (link). In this model we equate cost and edge length. At the time a connection is established, a shortest path connecting the pair of endpoints is selected. Of course only paths consisting of edges with sufficient unused bandwidth may be chosen.

Routing of multipoint connections may be modelled in a similar way. In the multipoint problem we wish to connect a set $D \subset V$. Instead of the shortest path, one is interested in the shortest subtree which contains the set D . Finding the shortest subtree connecting a set of points is a classical problem in graph theory known as the Steiner tree problem in graphs [19]. This problem has been shown to be NP-complete by Karp [40] in 1972. Consequently one is forced to consider approximation algorithms which are not guaranteed to produce optimal solutions.

7.1. Approximation Algorithms

There are several polynomial time approximations algorithms for solving the Steiner tree problem, which we have used as a starting point for work on multipoint routing. The *minimum spanning tree heuristic* (MST) [34] is probably the best known approximation algorithm. It can be shown that a solution produced by this algorithm will have cost that is never worse than twice that of an optimal solution. Our

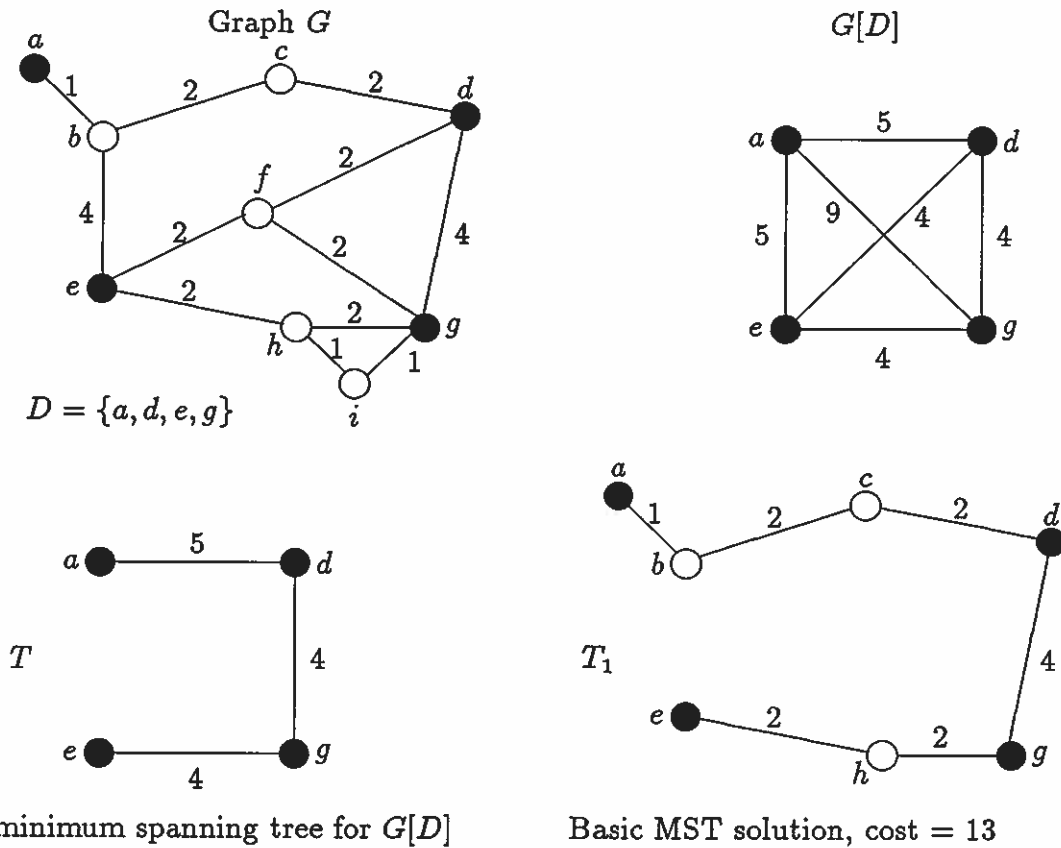


Figure 7.1: An Example of the Application of MST

experimental results indicate that MST typically yields solutions that are within five percent of optimal. Figure 7.1 illustrates an example of the application of MST. Here we are asked to connect the set of four nodes $D = \{a, d, e, g\}$. The first step of the algorithm involves constructing the derived graph $G[D]$. This graph is a complete graph on the four nodes in D , where the length of each edge corresponds to the length of the shortest path in the original graph G . The second step involves finding a minimum spanning tree for $G[D]$. This can be done using one of several polynomial time algorithms. Finally the edges of the minimum spanning tree for $G[D]$ are mapped back to paths in the original graph, taking advantage of path overlap. Note that the solution here has cost two units more than optimal.

To gain some insight into the probable performance of MST we have implemented two versions of this heuristic using the C++ language. The first version we refer to as the basic MST heuristic and is the algorithm described in the previous paragraph. We refer to the second version as the improved MST heuristic.

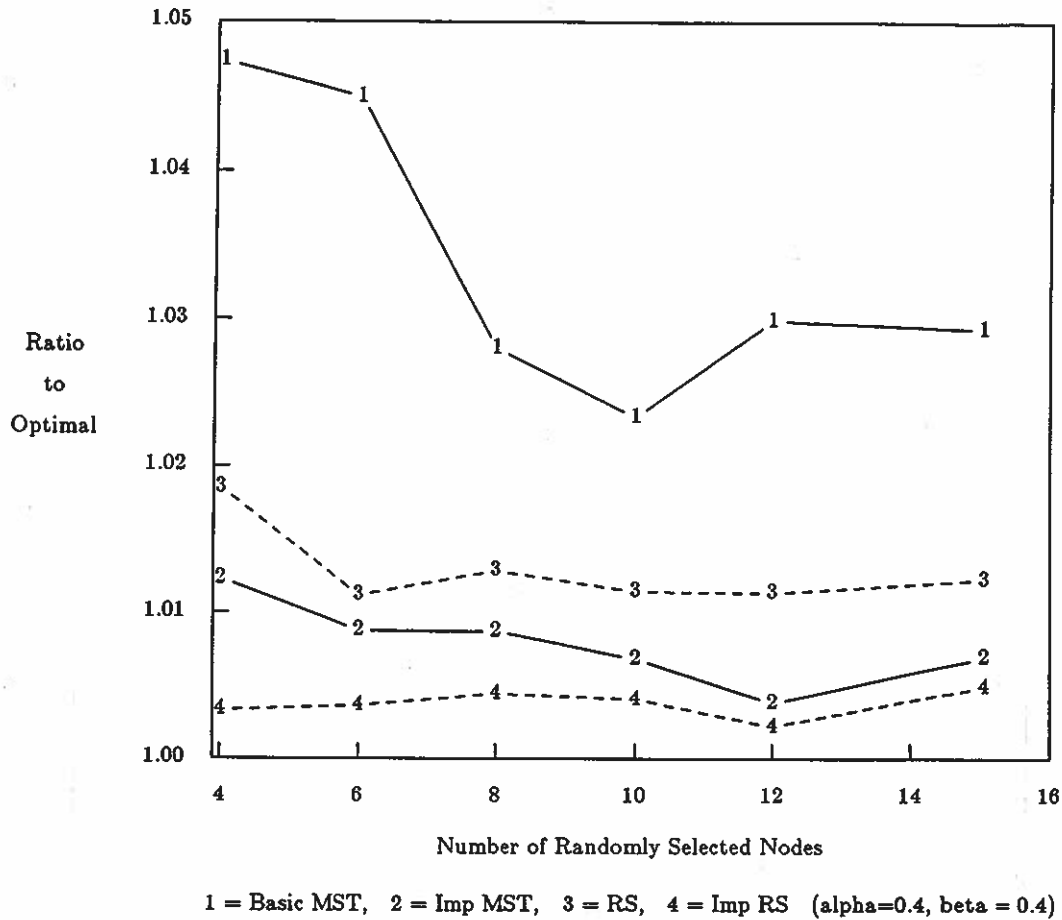


Figure 7.2: Experimental Performance of MST and RS

Improved MST takes the results of basic MST and forms a set D_1 which contains all of the nodes of D plus any additional intermediate nodes used in the solution produced by basic MST. Basic MST is then applied to the set D_1 . From the resulting tree any branch which does not contain nodes in D is pruned. Improved MST will always yield a solution that is at least as good as the solution produced by basic MST, but in the worst case will do no better.

We have also investigated a second approximation algorithm due to Rayward-Smith [47], which we refer to as RS. This algorithm addresses the one major problem of MST. MST considers only the distance between pairs of nodes to be interconnected and does not give any consideration to the importance of intermediate nodes. Our improved MST heuristic does consider intermediate nodes, but not in a systematic way. Thus improved MST does not have worst case performance any better than that of basic MST. On the other hand, RS makes an attempt to

choose intermediate nodes in a systematic way, based on a collection of functions $f_\ell: V \rightarrow \mathbb{R}^+$. At each stage ℓ of this algorithm RS chooses a path through a node for which the function f_ℓ has a minimum value. We have proved the the worst case performance of RS is within twice optimal and have found a class of problems for which the performance approaches $3/2$ optimal. It is apparent from considering a number of simple examples that RS should do better than MST in most cases. We had conjectured that the worst case performance was actually within $3/2$ of optimal but have recently found that it can be as bad as 1.61 times optimal. If the worst case performance of RS is actually bounded by a value that is less than twice optimal, this leads to the interesting possibility of modifying RS to yield even better worst case results while still maintaining polynomial time complexity.

For the purpose of running these experiments we have developed two random graph models RG1 and RG2, which have some of the characteristics of an actual network. In RG1 n nodes are randomly distributed over a rectangular coordinate grid. Each node is placed at a location with integer coordinates. In RG2 for each pair of nodes a random distance in $(0, L]$ is chosen. Then in both models edges are introduced between pairs of nodes u, v with a probability that depends on the distance between them. In the case of RG1 the Euclidean distance between each pair is used. The probability function is given by

$$P(\{u, v\}) = \beta \exp \frac{-d(u, v)}{\alpha L}$$

where $d(u, v)$ is the distance between nodes u and v , L is the maximum distance between a pair of nodes, and α and β are parameters in the range $(0, 1)$. The size of β determines the overall edge probability independent of distance, while small values of α decrease the density of longer edges relative to shorter ones. Finally the length or cost of each edge is set equal to the distance between its endpoints.

The experimental results of applying MST and RS to the graph model RG1 are illustrated in Figure 7.2. These experiments were run on five different twenty five node random graphs. Each experiment consists of choosing a subset of the graph nodes at random and then executing the basic and improved versions of both MST and RS. In addition an exact algorithm was used to determine the cost of an optimal solution. Each data point represents a set of twenty five experiments, five on each of five graphs. The graph displays the average ratio of the cost of the solution given by MST and RS to an optimal solution. It is interesting to note that results of experiments run with random graph model RG2 are not significantly different from those with RG1. The fact that RG1 introduces the Euclidean metric on the distances apparently makes little difference in the performance of these algorithms.

7.2. The Dynamic Steiner Tree Problem

The algorithms just presented assume that the problem is basically static and can be solved in a centralized fashion. In a large network one must rely on distributed algorithms in which no individual processor has global knowledge. In addition one cannot expect to know in advance all of the nodes that will be in a given connection. Thus, in their present form the MST and RS heuristics are most useful as tools against which the performance of more realistic algorithms can be measured.

We now consider the problem of the dynamic multipoint problem in which nodes are allowed to join or leave a connection at any point during the lifetime of that connection. Probably the simplest algorithm for handling dynamic multipoint routing is one which we refer to as the greedy algorithm. This greedy algorithm adds new endpoints to a connection by using the shortest path from the endpoint to a node already in the connection. Nodes are removed from the connection by deleting that branch which serves only the node to be removed. Note that if a given node is an endpoint in a connection as well as an intermediate node in a path to other nodes, removing it as an end point will not allow us to delete it from the connection. Figure 7.3 illustrates a sequence of five events handled by the greedy algorithm. Note that event four gives an example of a situation where the connection is not optimal. In fact this connection has a cost of three units more than that of an optimal solution.

We have implemented this greedy algorithm in C++ and have run a set of experiments on 195 node graphs, again using our random graph model RG1. For these experiments we used a simple probability model to determine if an event should be an addition or deletion of a node from a connection. The function

$$P_C(k) = \frac{\alpha(n - k)}{\alpha(n - k) + (1 - \alpha)k}$$

was defined for this purpose. Here P_C is the probability that an event is an addition of a node, k is the number of nodes in the current connection, n is the total number of nodes in the network and α is a parameter in $(0, 1)$. The value of α determines the fraction of nodes in the connection at equilibrium. In other words αn is roughly the average number of nodes in the connection. For example, when $k/n = \alpha$, $P_C(k) = 1/2$.

Figure 7.4 shows the results from one experiment. The figure contains two curves, the bottom one showing the number of endpoints in the connection as the experiment progressed and the top one showing the ratio of the length of the solution produced by the greedy algorithm to the length of the MST solution at each point in time. (This experiment was run on a graph with 195 nodes and with $\alpha = 6/195$.)

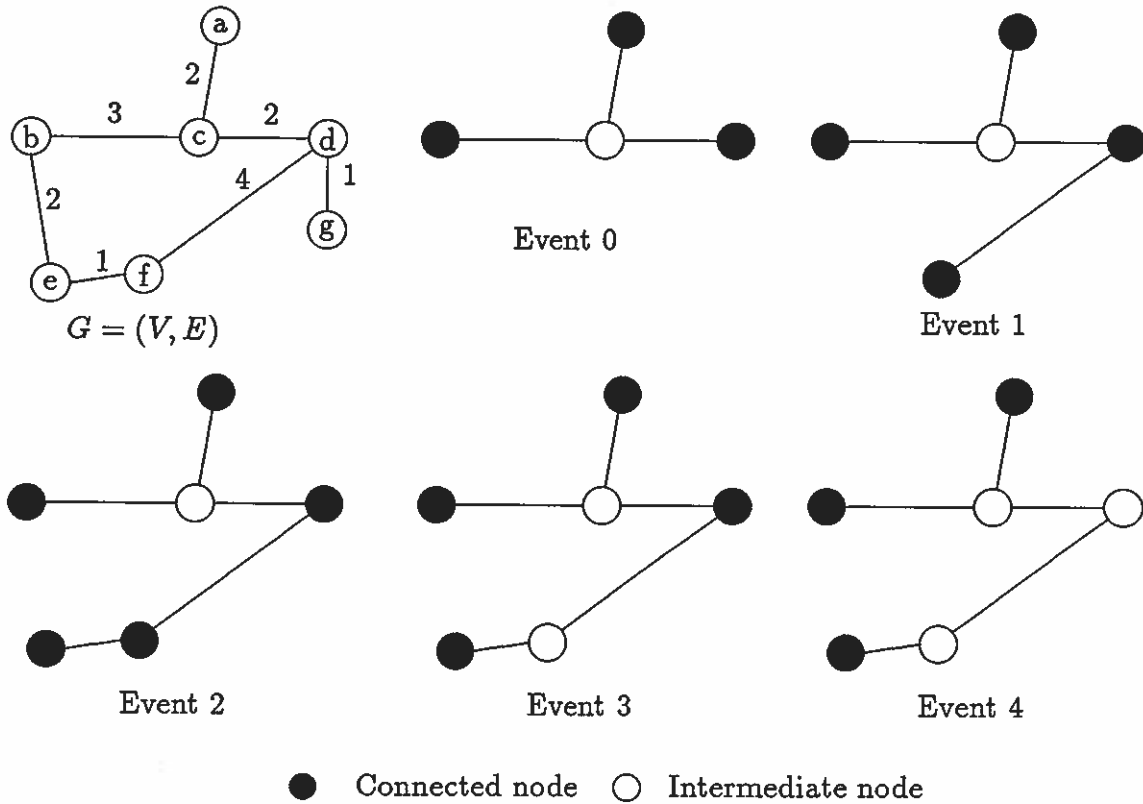


Figure 7.3: Dynamic Greedy Algorithm with Sequence: $a, b, d, f, e, \bar{f}, \bar{d}$

These experimental results indicate that the greedy algorithm does reasonably well in comparison to MST. On average the performance of the greedy algorithm is within 35% of MST. It does poorly on a few isolated events, but notice that this poor performance occurs at times when several nodes leave the connection; that is, it is not caused by bad routing decisions, but by the inability of the greedy algorithm to reconfigure a connection after its been reduced in size. There are several techniques that might be used to reduce the sharp deterioration in these cases, at the cost of worsening the performance slightly in other cases. For example, the choice of a connection path for a new endpoint could be based on function of both the length of the path to a node in the connection and the distance of that node from the center of the current connection.

Recently we have implemented a version of the greedy algorithm where the choice of a connection path is weighted by the distance of the connection point from the owner of the connection. Here the owner o is just a distinguished node chosen at random which cannot be removed during the life of the connection. In

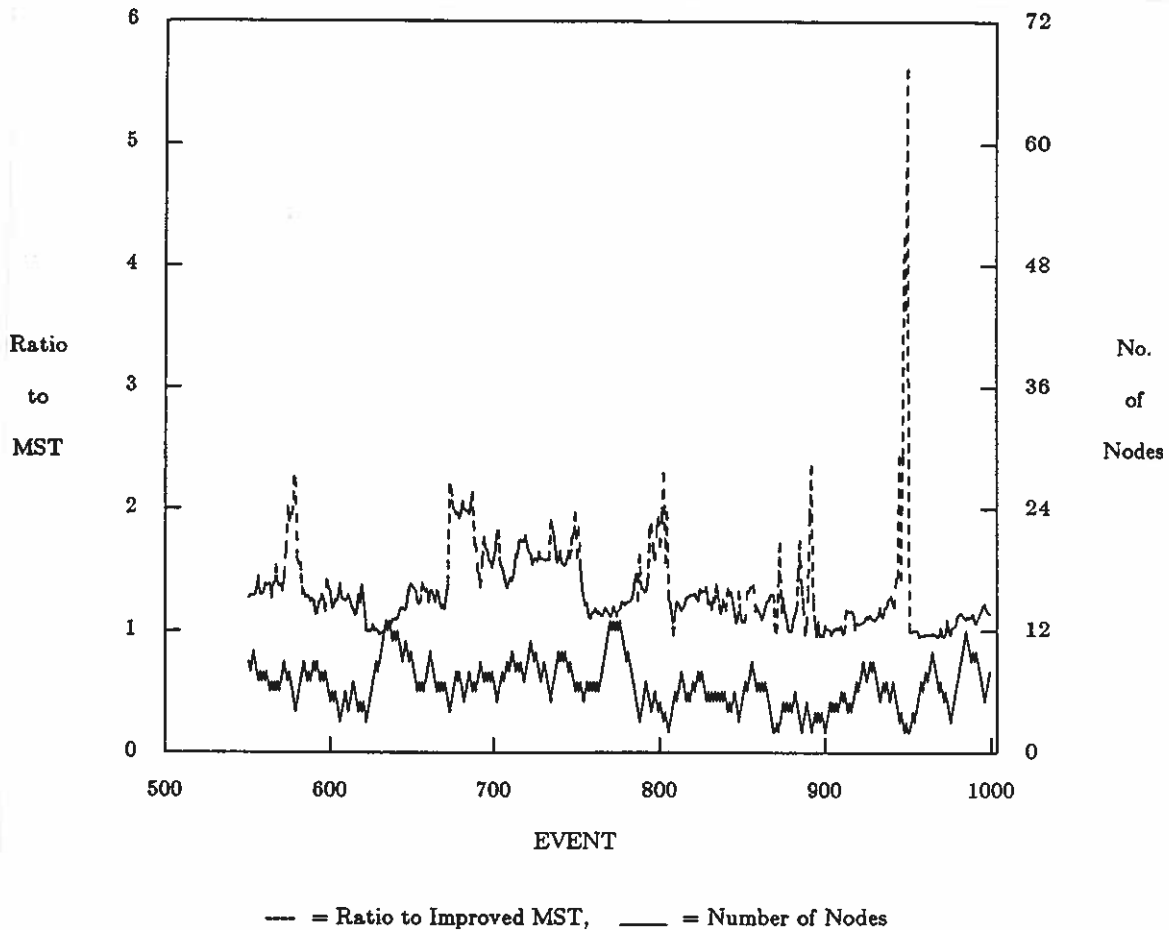


Figure 7.4: Experimental Results for the Greedy Algorithm

this version, greedy adds a node u by choosing a node v , in the connection, which minimizes the function W

$$W(u, v) = (1 - \omega)d(u, v) + \omega d(v, o)$$

where $0 \leq \omega \leq 0.5$. When $\omega = 0$ this version is equivalent to our basic greedy algorithm. If $\omega = 0.5$ a node u is added to the connection by the shortest path to the owner o . Our experimental results indicate that a value for ω in the neighborhood of 0.3 yields the best results. For example in one experiment worst event results were reduced from 3.5 with the original algorithm to just under 2.0 with a weighted algorithm using $\omega = 0.3$. Though the improvement in average performance was not as dramatic, in most cases we saw improvements of better than 5%

7.3. Random Graphs and Probable Performance

We have investigated the relationship between the values of α , β and the property of being connected for RG1 and RG2. To simplify our analysis we modified RG1 so that the nodes are placed on a sphere of radius 1. Here the maximum distance between two nodes is π . Using the results of Erdős and Rényi [15] we have derived the threshold functions for both models. To illustrate we have the function

$$\beta = \frac{2C(\pi^2\alpha^2 + 1) \ln n}{\pi^2\alpha^2(e^{-1/\alpha} + 1) n}$$

for model RG1. When $C > 1$ graphs will almost always be connected and when $C < 1$ the graphs will almost always be disconnected. In other words, as the number of nodes n grows the probability that a graph from RG1 is connected (disconnected) goes to 1 if $C > 1$ ($C < 1$).

We have looked at a simpler model of a random graph in which all edges have unit cost and the probability of an edge is uniform for all pairs of nodes. For a constant edge probability we know that almost all graphs have a diameter of 2. As a result, we have been able to show that for a small subset of graph nodes S , RS will almost always find the optimal solution while MST will not always find an optimal solution with this model. For example with an edge probability of 0.2 and $|S| = 10$ MST is expected to find an optimal solution approximately 50% of the time.

Most recently we have been looking at the diameter of random graphs where the edge probability is a decreasing function of n and is not bounded below by any positive value. We know from the work of B. Bollobas [4] that if the edge probability is given by

$$p(n) = \frac{(2Cn \ln n)^{1/d}}{n}$$

then almost all graphs have diameter d for $C > 1$. We can also show that for a graph G with this edge probability that the distance between a pair of nodes is almost always d . Thus for a small subset S of nodes in G the expected cost of a connection produced by the MST heuristic will be $d(|S| - 1)$. We still need to consider the expected cost of an optimal solution and the cost of a solution produced by RS. We suppose that both of these will be less than the cost of the MST solution.

7.4. Distributed Routing Algorithms

We now briefly consider the problems involved with the development of a distributed algorithm without access to complete global information. Most of the

work in this area deals with point-to-point networks. Algorithms have been developed which perform such functions as updating routing tables and finding shortest paths in a distributed fashion. For example the ARPANET makes use of distributed packet routing algorithms. Unfortunately most of the work with distributed routing algorithms cannot be applied directly to a network which handles multipoint connections. Thus most of the existing results will serve primarily as a starting point for the development of new algorithms.

The work of J.M. Jaffe [29] is of particular interest. He has considered the problem of multipoint routing with limited information. His work considers the effects of limited information on the Steiner tree problem. He shows given certain assumptions that it is not possible for any algorithm to produce optimal solutions in all cases. For example, Jaffe presents a class of algorithms which are restricted to *local information* and shows that the worst case performance for these algorithms can be no better than $2k/3$, where k is the number of nodes to be connected. These results indicate, that with limited data storage at each node, we are not likely to find a routing algorithm which yields optimal routing. In fact Jaffe's results lead us to believe that any distributed algorithm based on the quantity of data, which can reasonably be assumed to be available for local processing in a very large network, is not likely to have worst case performance even within a constant factor of optimal. On the other hand, Jaffe's results do not tell us anything about average performance, which is of great importance for a real network. Finally these results suggest that the performance of the greedy algorithm is reasonably good, and thus the greedy algorithm seems to be a reasonable starting point for the development of distributed multipoint routing.

8. Bandwidth and Buffer Management

Faculty
Research Associate
Graduate Students

Jonathan Turner
Riccardo Melen
Shahid Akhtar

One of the principal advantages of packet switching is its ability to support communication channels of any rate across a potentially wide range. Not only can different channels operate at different rates, but the rates of individual channels may vary over time. This latter property leads to the possibility of overload since there may be periods when the total offered traffic exceeds the network's capacity.

In conventional, low speed packet networks, such overload periods are controlled using a variety of feed-back oriented techniques, which attempt to detect overload and then apply control mechanisms that reduce the offered load. A common approach is to allow transmission of a packet from one switch to another only when the receiving switch is known to have a buffer available. During overload periods, such networks become congested with traffic backing up toward the sources, which are ultimately forced to reduce their rate of transmission until the congestion clears. This technique works well in networks with low speed or physically short transmission links. It works less well in networks with high speed links connecting switches that are separated by large geographic distances. The fundamental reason is that many packets (hundreds or thousands) can be in transit across a long, high speed link at any instant in time. With conventional data link protocols, buffers to store each of these packets are required in the receiving switch even though under normal conditions, only a few of these packets will be present in the switch at the same time. This leads to unreasonably high buffer requirements. Consequently, high speed packet networks use protocols that do not preallocate buffers. Instead, they simply permit packets to arrive in a relatively unconstrained fashion, and discard packets if insufficient buffer space is available. To keep the frequency of packet loss at an acceptable level, connections are allocated a portion of link bandwidth based on their traffic characteristics.

A key problem in the design of fast packet networks is how to perform this bandwidth allocation. This in turn depends on the behavior of information sources that may be very bursty. We are attempting to characterize the behavior of bursty sources and describe some of this work in section 8.1 below. Section 8.2 briefly describes how the analysis of 8.1 can be used to determine a connection's *effective bandwidth*, along with a practical method for calculating effective bandwidth in an operational switching system. Section 8.3 describes how user-specified bandwidth requirements can be enforced by the network. Section 8.4 introduces the additional problems that arise in multipoint connections and proposes a promising candidate solution. Section 8.5 describes a rather different approach to handling multipoint connections.

8.1. Queueing Behavior of Bursty Sources

We can model a bursty source as a two state Markov chain. When in the *idle* state, a source transmits no data and when in the *active* state, it transmits λ packets per second. Sources that make infrequent transitions between the active and idle states are called *bursty*. When a bursty source becomes active it stays active for a relatively long period of time. We define the *burst factor* B of a source to be the average time spent in the active state, times the difference between the source's peak and average rates.

We can model the behavior of a queue of length n receiving traffic from m independent and identical bursty sources, as a finite Markov chain with states $s_{i,j}$, $1 \leq i \leq m$, $1 \leq j \leq n$. We interpret state $s_{i,j}$ to mean that i sources are in their active state and j packets are in the buffer. This model is illustrated in Figure 8.1. This Markov chain can be solved numerically to determine the state probabilities and from these the, fraction of transmitted packets lost due to queue overflows. We show two plots obtained in this way in Figure 8.2. These plots show packet loss rates for a queue of length $n = 64$, fed by $m = 64$ sources. In the plot on the left, the peak rate λ is held constant at $1/16$, while the offered load (ρ) and burst factor B are varied. In the plot on the right, the burst factor is held constant at 4, while the offered load and peak rate are varied. In both plots, we note packet loss rates substantially higher than predicted by an M/M/1 queueing model. Even for a burst factor of 1, our target loss rate of 10^{-6} is achieved at an offered load of about 70%, whereas an M/M/1 model predicts this loss rate at an offered load of about 83%. Increasing either the burst factor or the peak rate increases the loss rate substantially, requiring a corresponding decrease in the allowable offered load.

The Markov chain model is the simplest analytical model that captures the essential characteristics of bursty information sources. We have also used simula-

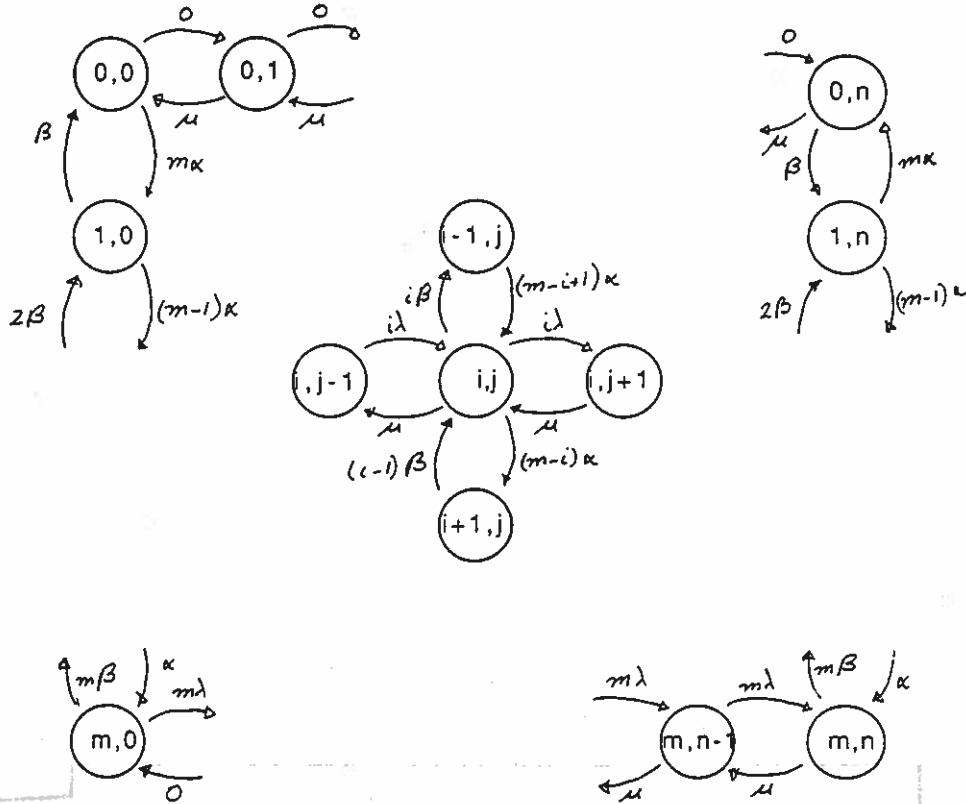


Figure 8.1: Markov Chain Model

tion models to explore the behavior of queues fed by bursty sources. Using such simulation models it is possible to study sources with more complex behavior. The drawback of simulation is that it does not allow accurate determination of the extremely small packet loss rates of interest. Details of these simulations will appear in a later report.

The Markov chain model can be extended in a couple different ways. First, it is possible to model the behavior of a queue carrying traffic from several types of sources. That is, two state sources with different peak rates, average rates, and burst factors. If there are k types of sources, the resulting Markov chain has $k + 1$ dimensions. Unfortunately, the number of states in the model increases rapidly with the number of types of sources and the time required to obtain accurate numerical solutions increases also. With more efficient algorithms, we expect to be able to solve models with two or three types of sources, but that is probably the practical limit of this particular method.

The model can be extended in another way also. Instead of two state sources, we can consider k state sources which transmit at different rates depending on their state. A Markov chain modelling an n slot queue fed by m such sources has k dimensions. Again, the number of states grows rapidly with k limiting the

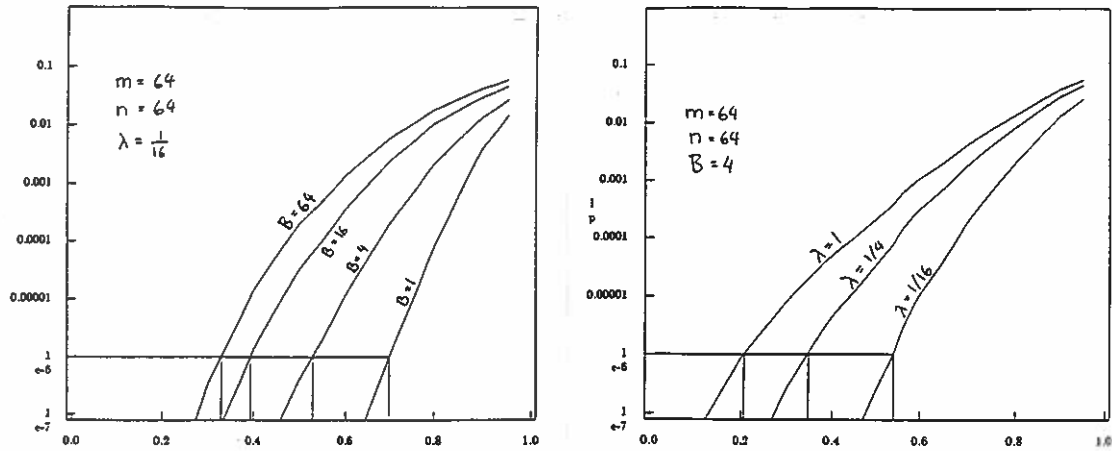


Figure 8.2: Packet Loss Rates

usefulness of this method to sources with just a few states.

8.2. Bandwidth Allocation

The bandwidth allocation problem in fast packet networks is to determine if a given set of connections with known traffic characteristics can share a link with acceptable packet loss rate. If the sources can be adequately described by two state Markov chains, this problem can be solved in principle by the methods of the previous section. Unfortunately, the time required for solving a Markov chain model with many different source types is prohibitive, particularly in the context of a practical communications network, where the time available to make such a decision is on the order of 10 ms. An interesting research problem would be to develop methods of synthesizing models with far fewer states that would yield approximate solutions of reasonable accuracy. We take a simpler approach here, using Markov chain models for a single source type to compute an *effective bandwidth*, then using the effective bandwidth as the basis for bandwidth allocation decisions.

Consider a bursty source with a peak bandwidth of λ packets per second, an average rate of ρ packets per second and a burst factor of B . Let $L(\lambda, \rho, B, m, n)$ be the packet loss rate that results when m such sources feed an n slot queue. Let L^* be the largest packet loss rate we wish to allow in the system and let m^* be the largest value of m for which $L(\lambda, \rho, B, m, n) \leq L^*$. We define the *effective*

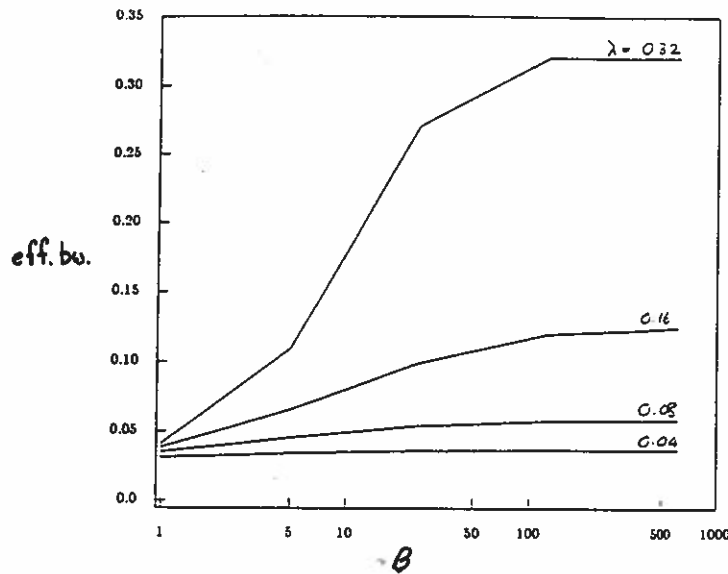


Figure 8.3: Effective Bandwidth

bandwidth of the source by

$$f(\lambda, \rho, B, n) = \left[m^* + \frac{L^* - L(\lambda, \rho, B, m^*, n)}{L(\lambda, \rho, B, m^* + 1, n) - L(\lambda, \rho, B, m^*, n)} \right]^{-1}$$

Using the Markov chain analysis of the previous section, it is reasonably straightforward to compute the effective bandwidth. Figure 8.3 displays a plot of effective bandwidth for sources with an average bandwidth of about 0.03 and various values of the peak bandwidth and burst factor. The number of buffer slots used here is 32 and the target loss rate is 10^{-5} . Note that for small to moderate peak bandwidths, the effective bandwidth approaches a maximum somewhat lower than the peak, but for large peak bandwidths and burst factors of 100 or more, the effective bandwidth approaches the peak.

In an operational switching system, it is probably impractical to compute the effective bandwidth of a source every time a connection is established, since the time to solve the Markov chain model can be substantial. On the other hand it is reasonable to precompute a table of effective bandwidths for a wide variety of sources and then use interpolation to determine values that do not appear in the table. Since the values of n and L^* are fixed for a given switch, such a table has three dimensions. We estimate that about twenty values in each dimension, covering three orders of magnitude should suffice in practice. This would yield a table of 8,000 values. We are currently evaluating the accuracy of the interpolation scheme for such a table and will report on that in a subsequent report.

We close by noting that this approach to bandwidth allocation may result in decisions that lead to higher than expected packet loss rates. The actual behavior

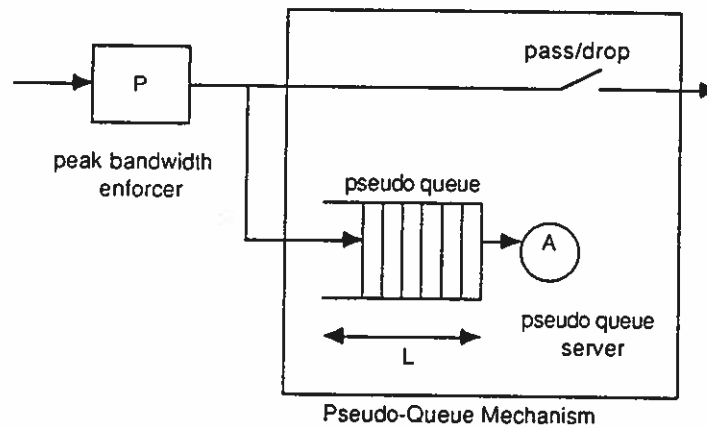


Figure 8.4: Simple Bandwidth Enforcement Mechanism

of a queueing system depends on the particular combination of sources feeding the queue. We have ignored that here, computing effective bandwidth based on the behavior of identical sources, not an arbitrary mixture. Consequently, decisions based on this method can err either on the liberal or conservative side, leading to excessive packet loss in one case and excessive connection blocking in the other. This is inherent in any method that relies on a single number to characterize the behavior of a bursty source, and unfortunately such methods are the only kind that appear computationally tractable. There is a slightly different approach that may yield better results. Rather than compute effective bandwidths for identical sources, we could compute them in the context of some representative background load. Possibly several different background loads could be used, and the appropriate one selected based on actual source statistics.

8.3. Bandwidth Specification and Enforcement

The bandwidth allocation methods discussed in the previous section, all require that the user provide some description of the connection's traffic characteristics, when the connection is established. This *bandwidth specification* must include the connection's *peak rate*, its *average rate* and *burst factor*. The network requires this information to determine the effective bandwidth. In addition, the network must have a straightforward way of preventing the user from sending traffic at a higher rate than allowed by his specification. This is done by having the network monitor the connection's traffic as it enters the network and discard any packets that exceed the rate spec. We refer to this enforcement mechanism as a *traffic valve*. (Alternatively, the valve could reset the priority of the packet to a lower

value and allow it to proceed, or buffer the packet and flow control the user across the access link.)

One simple way to implement a traffic valve is called the *pseudo-buffer* mechanism. The network monitors the flow of traffic entering the connection and simulates the effect of this traffic on an imaginary buffer. When the user sends a packet, the length of the pseudo-buffer is increased by one. So long as this doesn't cause the pseudo-buffer to overflow, the packet simply passes over the connection; if the pseudo-buffer does overflow, the packet is discarded. This idea is illustrated in Figure 8.4. The pseudo-buffer accepts packets at a certain maximum rate (the connection's peak rate), is drained at a constant rate (the connection's average rate) and has a maximum length (the connection's burst factor). These three parameters may all be specified by the user, allowing him in effect, to specify the characteristics of a *virtual private link*.

A pseudo-buffer mechanism that simultaneously monitors all the logical channels on a single access link can be implemented in hardware in a straight-forward fashion. The main component is a memory that contains the values of the parameters associated with each pseudo-buffer, the state of the pseudo-buffer at the time it was last updated and a time field indicating when its state was last updated. Whenever a packet arrives on the link, the packet's logical channel number is used to select the appropriate set of parameters from the memory, the parameters are updated and written back to memory and a decision is made as to whether or not to discard the packet. The amount of information required for each channel is sixteen bytes, implying that 512 channels can be monitored using 64K bits of memory.

8.4. Multipoint Congestion Control

The previous sections deal mainly with point-to-point connections and multipoint connections with a single transmitter. In these cases, it is straightforward to provide the bandwidth enforcement function entirely at the edge of the network. Connections with two or more transmitters and three or more receivers raise new issues, because in this case it becomes possible for packet streams from different transmitters to converge with one another inside the network, creating loads larger than are permitted at the boundary.

When one considers applications involving multiple transmitters, it becomes evident that often, while there are many *potential transmitters* there is only one or a few *concurrent transmitters*. The solution proposed here involves adding some functionality to the internal switch nodes that controls access to buffer slots within switches. The underlying idea is to allocate slots in link buffers to connections in

proportion to the allocation of bandwidth on the link. For example, if a given connection has been allocated say 20% of the bandwidth on a given link, then it is also allocated 20% of the buffer slots in that link's buffer. This allocation has no effect unless the link buffer becomes full; in the event that it does become full, the buffer allocation is used to determine which packets should be discarded. That is, connections using more than their allocated buffer slots lose packets during overload while connections that are operating within their allocation are protected.

We now give a more precise description of this policy and describe a simple mechanism to implement it. Suppose we have a set of connections c_1, \dots, c_m sharing a common transmission link. Each connection c_i has an associated *rate* r_i . The link has a rate R and the link buffer has a capacity of B packets. We require that $\sum r_i \leq R$, but note that this does not imply that each source transmits at a constant rate of r_i , but rather that r_i is the amount of link bandwidth reserved for the connection during overload periods. We define $b_i = \lceil Br_i/R \rceil$ to be the number of buffer slots allocated to c_i .

During overload periods, the link buffer fills and overflows (we can take this as the definition of overload). The objective of buffer management is to discard packets selectively so that ideally, only connections c_i using more than "their share" of the link buffer are affected by the overload. We propose to do this by supplementing the link hardware with a table containing an entry for each connection. The entry for c_i consists of two fields, one containing b_i , the number of buffer slots allocated to c_i , and the other containing β_i , the number of buffer slots currently being used by c_i . Whenever a packet on c_i is received, the corresponding table entry is read and β_i is incremented, then written back to memory. If $\beta_i > b_i$ the packet is marked as an *excess packet*, meaning that it may be discarded if an overload occurs. When a packet on c_i is removed from the buffer (either to be transmitted or discarded) β_i is decremented. This mechanism is illustrated in Figure 8.5.

One simple way to implement the buffer is as a priority queue with the excess packets being treated as lower priority. One drawback of this implementation is that it introduces the potential for packets in a given connection to pass one another. A better solution is a buffer that maintains packet ordering, but still allows excess packets to be discarded first in an overload. Figure 8.6 shows a block diagram of such a buffer. The packets are stored in the memory array at the bottom of the figure. They enter on the left, passing through an input shift register (ISR) before being strobed into memory. In the simplest case, each packet occupies one full column of the memory array, although this isn't essential. Packets leave on the right, passing first through an output shift register (OSR). A master control circuit provides global control logic and timing.

The status block at the center maintains information about the packets stored in each column of the memory array. It supports writing to and reading from the

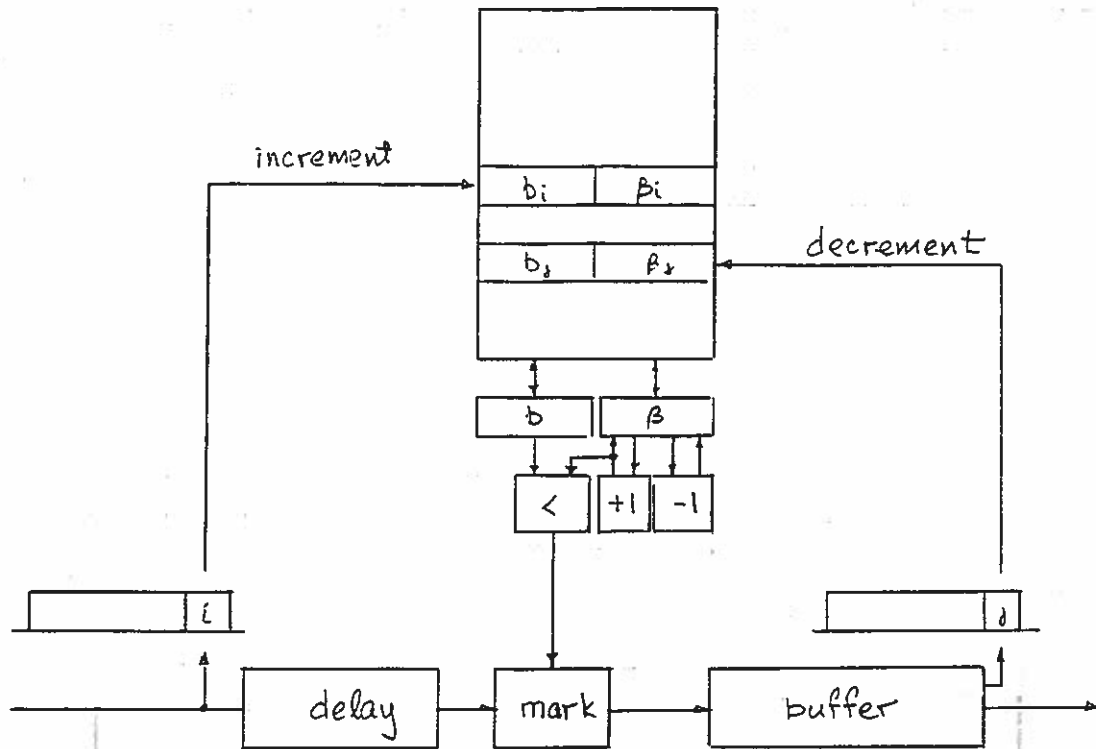


Figure 8.5: Buffer Management Mechanism

buffer in FIFO order. It also supports writing to a memory column occupied by an excess packet. It can be implemented as a one-dimensional array of control slices, with each slice corresponding to a column of memory. Each slice contains a *control word*, consisting of a *column number*, which identifies the column controlled by the slice, a *data present bit*, which is set if the column contains a packet, an *excess bit*, which is set if the packet in the column is an excess packet, and a *channel number*, which identifies the channel that the packet belongs to.

Notice that if $\sum b_i < B$, we can guarantee that only excess packets are lost. If such a guarantee is required, we can provide it by refusing to accept new connections that would violate this condition. We note however that there are common situations in which this strategy is overly restrictive. Suppose for example, we have 1000 connections with rates $r_i = .001$ that we would like to multiplex on a single link. To accommodate these connections subject to the constraint that $\sum b_i < B$ we need at least 1000 buffer slots, a number which may be impractically large. We note however that in this situation, we can tolerate much smaller buffer capacities since the probability of buffer overflow will generally be small, even using a buffer

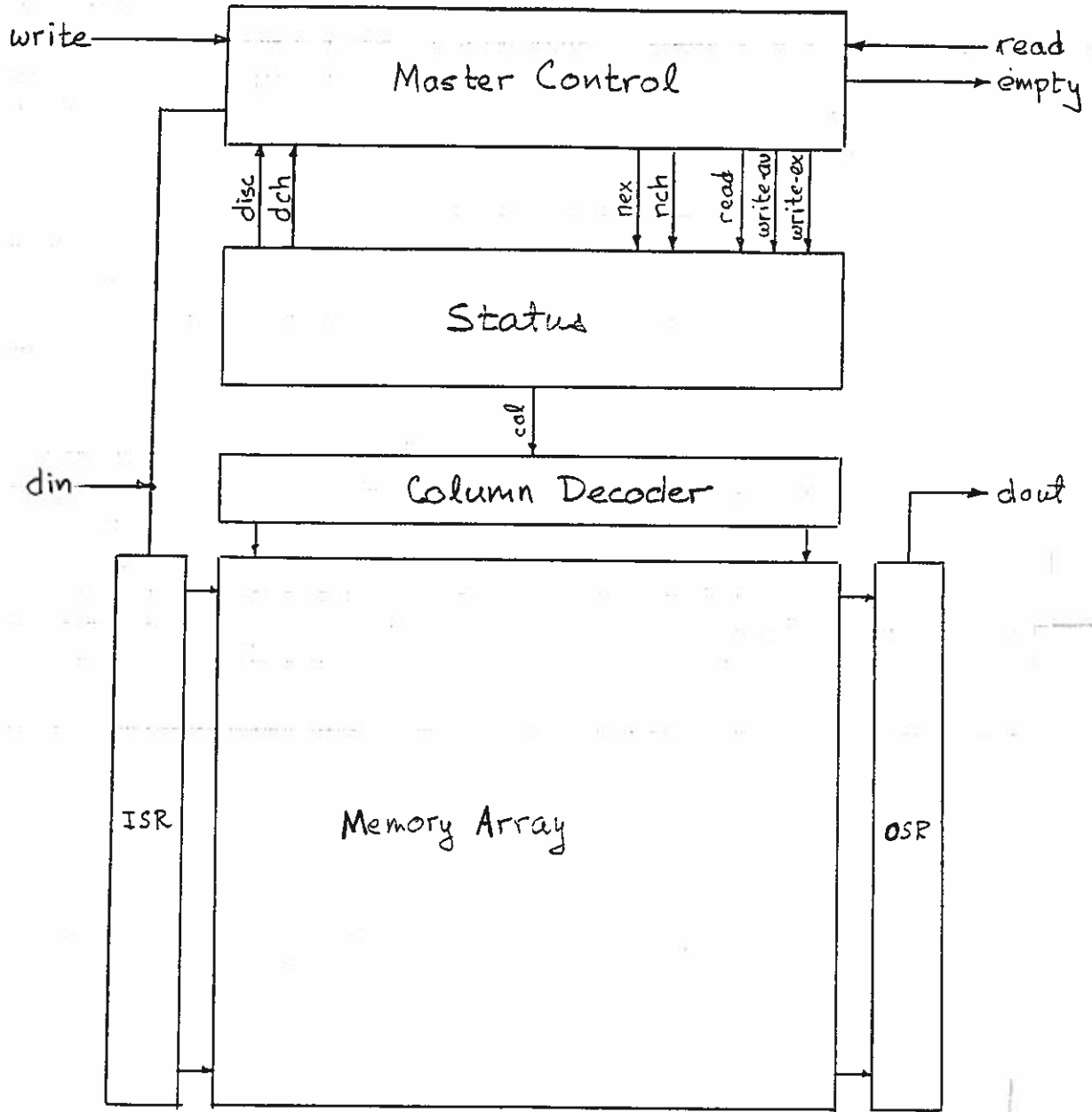


Figure 8.6: Buffer Implementation

with say 50 slots.

This mechanism can effectively control the bandwidth usage of multipoint connections as well as point-to-point connections. We require that the user provide a rate specification not just for a single transmitter but the for the total connection as well. Using this information, the network can compute an effective bandwidth, which in turn is used to determine the buffer allocation. There remains a potential for local overloads occurring in a switching system in the case of a multipoint connection with three or more links at a switch, two of which result in packets entering the switch. This local overload can be controlled by discarding packets before entry to the switch fabric if the transmit buffer at the incoming packet processor is in the overload state and the connection the packet belongs to is already using its share of that transmit buffer.

8.5. Access Arbitration in Multipoint Channels

In the previous sections, our objective has been to allocate bandwidth to users to get acceptable performance and then control each connection's ability to access bandwidth and buffers so that each gets its share. In the case of multipoint connections, this may involve discarding packets from a connection with several simultaneous transmitters, without regard to where the packets came from. In this section we consider the possibility of active arbitration among the set of transmitters in a multipoint connection. In this model, terminals in a multipoint connection transmit information in bursts consisting of many packets and compete for the right to transmit bursts. In the simplest case, the channel allows only one terminal to transmit at a time; this can be extended to k concurrent transmitters. The problem resembles contention resolution in local area networks. It is distinguished by the topology of the channel, the magnitude of the delays involved and the potential for multiple transmitters. In this section we summarize work that we have done on the access arbitration problem. A more detailed report can be found in [39].

Perhaps the most obvious approach to access arbitration in a tree-structured channel is to supply the connection with some number k of transmit permits or tokens, and require that a terminal possess at least one token before being allowed to transmit a packet. This approach limits the number of simultaneous transmitters to k , while allowing the set of transmitters to vary over time, through the passing of tokens. The network can allocate bandwidth for k simultaneous transmitters, independent of the total number of terminals in the connection.

While this strategy seems simple enough, finding a practical implementation for a high speed packet network is not as straightforward as it might appear. The reason is that passing of tokens must be completely reliable; since the underlying

network may lose packets on occasion, a protocol is required that allows tokens to be passed reliably, while at the same time preventing terminals from creating new tokens. We have devised two algorithms which take rather different approaches to the problem. The first provides a simple and practical solution that can be implemented using a set of distributed monitor processes at the access links connecting the terminals to the remainder of the channel. In this algorithm, the network plays a passive role, with the terminals handling most of the work associated with token passing, while the network provides minimal support for reliable transmission and prevents token creation by the terminals. In the second algorithm, the network plays a more active role, distributing tokens to users based on request messages; this approach, while more complex can reduce the latency associated with token passing.

In the first algorithm, terminals transmit two types of packets, *data* and *token* packets. Every token packet has two fields, one containing the *token id*, and another containing the *destination terminal*, that is the identity of the terminal that is to receive the token. The internal nodes of the channel, replicate all received packets and propagate them throughout the channel.

The algorithm is implemented by a collection of *monitor processes*, located at the terminals' access links. The monitor processes observe the flow of packets over the channels and are responsible for preventing a terminal from transmitting a packet unless it is in possession of a token. The monitor processes, also prevent creation of new tokens by making sure that terminals pass only those tokens that are in their possession. The monitors provide indirect support for reliable token transmission; if a terminal passes a token to another and determines that the token packet was lost, it is allowed to retransmit the packet. The tricky part, is allowing such retransmissions without introducing a mechanism that allows the user to create new tokens. We have devised an effective method for handling this problem which is described in detail in [39].

In the second algorithm the network plays a more active role in token distribution, so as to reduce the latencies associated with token acquisition. In the new algorithm, a terminal with data to send first requests a token, then waits for the network to provide one. The algorithm is implemented by two types of processes. The first is a monitor process, which observes the passage of token control messages and allows data packets to be sent only when the terminal is in possession of a token. The second type of process implements token circulation; there is one such process for each internal node in the connection. One of the internal nodes, is designated the *root* of the channel; all other nodes u in the tree have a unique *parent*, which is the neighboring node that lies on the path from u to the root. The root can be any node, but the best performance is obtained when it is at the center of the tree. Whenever there are no pending requests for tokens, unused

tokens propagate up the tree to the root. Token requests also propagate up the tree, but each node maintains a list of token requests from its subtrees and if a token is received from either the parent or a subtree, while a request is pending, that token is used to satisfy the request. This simple form of the algorithm can be improved upon in several respects; details can be found in [39].

The token-based approach described above is inherently limited in performance due to the time required for token acquisition. Better performance is possible by allowing terminals with data to send to simply transmit the data at will and allow the internal nodes to select one (or several) from a set of competing bursts. Several algorithms for this problem have been devised. We briefly describe one of them here.

For the contention-based algorithms, we assume that a burst consists of a start packet, one or more data packets and an end packets and that the start and end packets can be transmitted with perfect reliability (these assumptions can be relaxed, but for simplicity we will not do so here). All packets transmitted include the identity of the transmitting terminal. The maximum number of simultaneous transmitters is k . We provide priorities for each burst by including a priority field in each start packet. Once a burst becomes active, it can be interrupted only by a higher priority burst. If two bursts of the same priority contend with one another, an arbitrary decision is made to select the winner. This arbitrary decision is based on another field of the start packet which is filled with an arbitrary random value when the packet enters the network.

The key part of any contention-based arbitration algorithm is to ensure that the selection of which of a set of competing bursts should gain access to the channel is done in a consistent fashion. The approach we use here is to exploit the natural numeric ordering on the triples consisting of the priority, the random value and the terminal identifier. Each internal node monitors the traffic passing through it and if more than k bursts attempt to pass through at once, it ceases propagating one of the bursts. The burst that is discontinued is the one with the lowest priority, with ties broken using first the random value and then the terminal identity. Each node stores the values of these triples for all bursts that are active in a connection. This algorithm can be efficiently implemented in hardware in a practical multi-point communication network. The main hardware cost is for the memory, which amounts to roughly $5k$ to $10k$ bytes per channel.

9. Packet Video

Faculty
Graduate Student

Jonathan Turner
Shabbir Khakoo

The use of packet networks for carrying video signals raises a variety of new research issues. We have recently begun an investigation of some of these. In this section, we summarize our initial observations and outline our research objectives. We begin with a brief review of methods used to reduce the bandwidth required to transmit video signals. We feel such methods will become even more important in the coming era of relatively inexpensive bandwidth, since the combined effect of inexpensive bandwidth and inexpensive codecs will be to drastically expand the market for two-way video, and because we expect a growing interest in dramatically higher resolutions.

A variety of techniques including transform coding, motion compensation, differential coding and adaptive quantization are commonly used to reduce the bandwidth required for video signals. Motion compensation [43,44] is among the most powerful techniques. It exploits the fact that there is relatively little change in detail between adjacent frames of a sequence. Thus by only transmitting the change in detail referenced to an initially transmitted frame, a significant compression may be realized.

Typical video codecs combine several techniques to yield a hybrid; an example of such a hybrid codec is shown in Figure 9.1. In such a codec, a previous frames is stored in the frame buffer (FB). The current frame is divided into blocks of typically 16 by 16 pixels. The search unit (S) compares each block to the previous frame to find the position in the previous frame that most closely matches the block. The result of this search is a displacement vector for the block which is transmitted to the remote decoder. The displacement vector is also used to select a block from the frame buffer and then this block along with the current block are subjected to a two-dimensional discrete cosine transform (DCT) and then the differences between the corresponding pixels are computed. These differences are then quantized using an adaptive quantization scheme. Typically, this involves

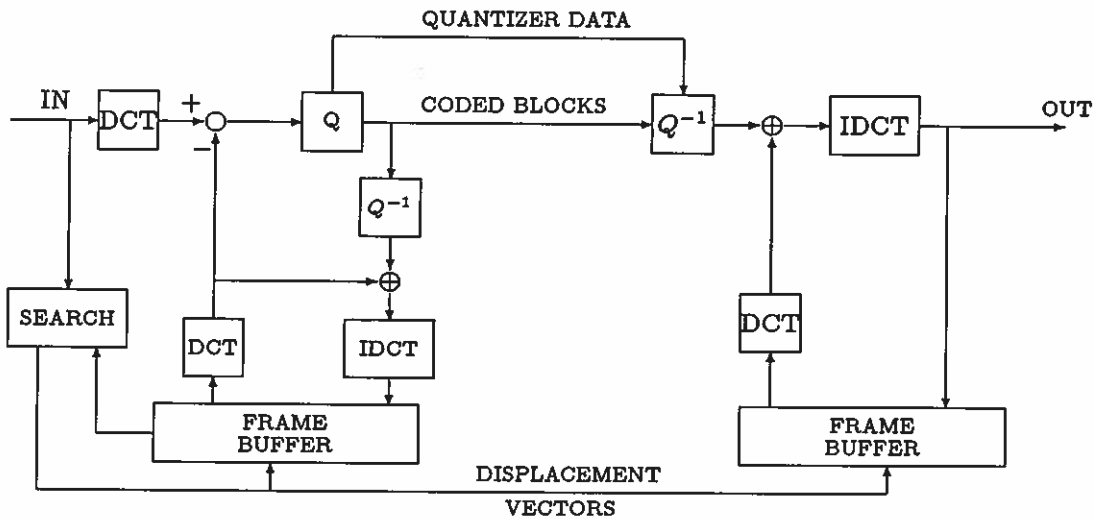


Figure 9.1: Hybrid Video Codec

classifying the block based on the amount of change and using one of several sets of quantization parameters. The coded blocks are then transmitted to the far end coder and the process reversed. At both coder and decoder, the decoded blocks are fed back into a reference frame buffer. Using such techniques, compression rates approaching 50:1 are possible with reasonable quality. Video conferencing codecs gain additional compression by starting with a relatively low resolution signal (256 by 256 is typical) and using a frame refresh rate of 7.5 frames per second. These techniques provide 16:1 compression over broadcast quality video.

When video coding is used in conjunction with a fixed rate communication channel, it is necessary to suppress the natural variability in the codec's bit rate. This is handled in two ways, as shown in Figure 9.2. First, a rate-equalizer buffer is placed between the output of the codec and the channel, so as to absorb short term variations in coding rate. During periods of rapid motion, the buffer can become full, necessitating a reduction in the coding rate. Several methods [43] of optimizing the feedback mechanism for maximal picture quality within a given bandwidth budget exist (typically these involve changing the criteria used by the quantizer), but large variations in quality are unavoidable, especially at high compression rates.

Packet switching systems, on the other hand, needn't constrain the codec to a constant bandwidth channel. Rather, the coding rate can vary in a relatively unconstrained fashion. Where the conventional codec attempts to maintain a constant or nearly constant data rate, while minimizing distortion subject to that constraint, a packet video codec can seek to minimize the number of bits per frame

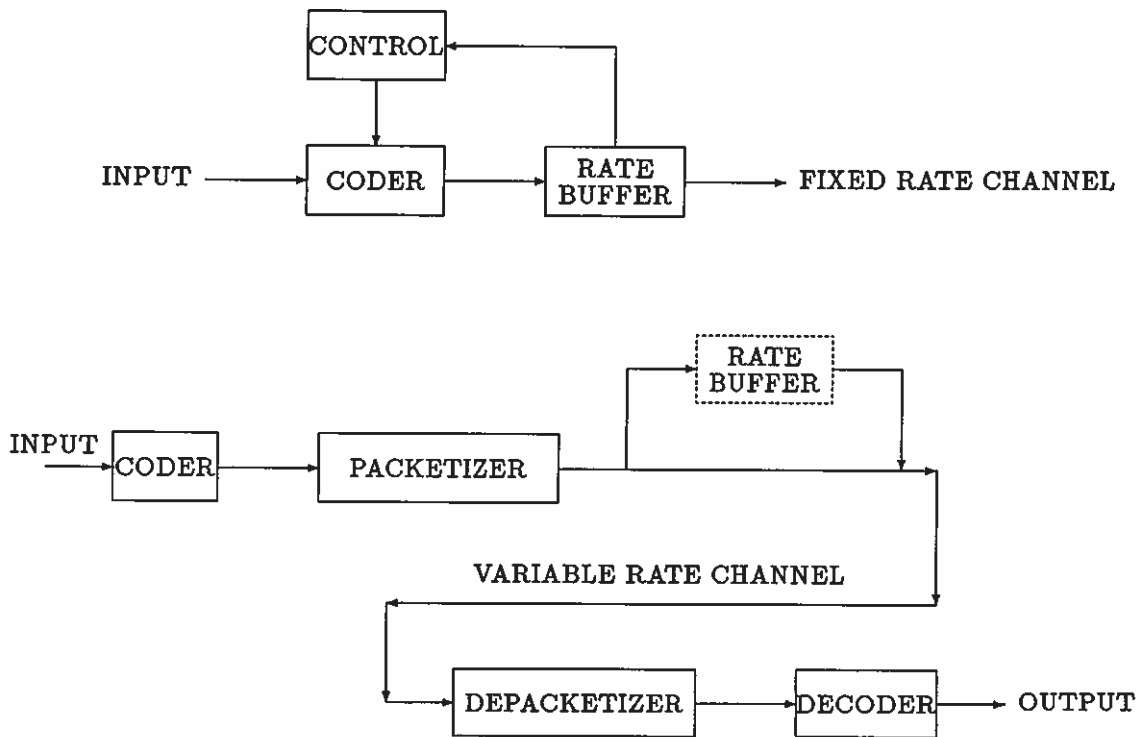


Figure 9.2: Fixed vs. Variable Rate Channels

subject to a constraint on distortion. The ability to transmit at high rates for short periods makes possible substantially better quality in periods of rapid motion.

The overall structure of a packet video system is shown in Figure 9.2. The data from the coder passes through a packetizer which organizes the data into packets with additional header information, then sends the packets through the network to the decoder which includes a depacketizer, which detects lost packets and restores packet order, based on sequence numbers provided by the packetizer. In this system, the rate buffer is optional, since the network can absorb the normal fluctuations in the coder's data rate. Since the network's bandwidth management mechanisms may constrain the coder's rate to some degree, a rate buffer may be of some value, but does not appear essential.

Packet loss can have a serious implications on the quality of the signal, particularly since it can result in a loss of synchronism between the coder and decoder. Since the round trip delay through the network is large, retransmission of packets is not the most efficient means of tackling the problem of packet loss. One simple method of overcoming this problem [65], is to replace the lost data by dummy

information that is derived either from the previous decoded picture or by interpolating between the received data of the current image under consideration. Either one of these method can reduce the effect of a lost packet substantially. At the same time, it is necessary to transmit redundant information at a low rate, to allow synchronization to be recovered.

Our initial investigations of packet video have been limited to studying existing video coding methods and developing an understanding of the implications of packet transport on these methods. We have begun to develop a set of programs that will allow us to apply some of these coding methods to actual video sequences. We plan to use these programs for several purposes. First, we wish to obtain a better understanding of the typical variation in data rates. Among other things, we need to understand the duration and intensity of high data rates during periods of rapid motion. Second, we will use these programs to explore various methods for coping with lost packets and measure the effect of various packet loss rates on the quality of the resulting signal. We also plan to study the implications of high resolution video on codec architecture. Currently, high compression rate codecs can operate with fairly limited parallelism because they use video signals of limited resolution and relatively low frame rates. The application of such techniques to higher quality video will require a greater reliance on parallelism and on custom integrated circuits to perform the necessary computations.

Bibliography

- [1] Agrawal, D. P. "Testing and Fault Tolerance of Multistage Interconnection Networks," *Computer*, 4/82.
- [2] Batcher, K. E. "Sorting Networks and Their Applications," *Proceedings of the Spring Joint Computer Conference*, 1968, 307-314.
- [3] Beckner, M. W., T. T. Lee, S. E. Minzer. "A Protocol and Prototype for Broadband Subscriber Access to ISDNs," *International Switching Symposium*, 3/87.
- [4] B. Bollobas. "The Diameter of Random Graphs," *Transactions of the American Mathematical Society*, 267:41-52, 9/81.
- [5] Bubenik, Richard. "Performance Evaluation of a Broadcast Packet Switch," Washington University Computer Science Department, MS thesis, 8/85.
- [6] Bubenik, Richard and Jonathan S. Turner. "Performance of a Broadcast Packet Switch." Washington University Computer Science Department, WUCS-86-10, 6/3/86.
- [7] Coudreuse, J. P. and M. Servel. "Asynchronous Time-Division Techniques: An Experimental Packet Network Integrating Videocommunication," *Proceedings of the International Switching Symposium*, 1984.
- [8] Coudreuse, J. P. and M. Servel. "Prelude: An Asynchronous Time-Division Switched Network," *International Communications Conference*, 1987.
- [9] Day, C., J. N. Giacomelli and J. Hickey. "Applications of Self-Routing Switches to LATA Fiber Optic Networks," *International Switching Symposium*, 1987.
- [10] De Prycker, M., and J. Bauwens. "A Switching Exchange for an Asynchronous Time Division Based Network," *International Communications Conference*, 1987.

- [11] Dias, D. M. and J. R. Jump, "Packet Switching Interconnection Networks For Modular Systems," *Computer*, vol. 14, no. 12, 12/81, 43-53
- [12] Dias, D. M. and J. R. Jump, "Analysis and Simulation of Buffered Delta Networks," *IEEE Transactions on Computers*, vol. C-30, no. 4, 4/81, 273-282
- [13] Dias, D. M. and Manoj Kumar. "Packet Switching in $N \log N$ Multistage Networks," *Proceedings of Globecom 84*, 12/84, 114-120.
- [14] Dieudonne, M. and M. Quinquis. "Switching Techniques Review for Asynchronous Time Division Multiplexing," *International Switching Symposium*, 3/87.
- [15] P. Erdos and A. Renyi. "On random graphs I," *Publication Mathematica Debrecen*, 6:290-297, 1959.
- [16] Feng, Tse-yun and Chuan-lin Wu, "Fault-Diagnosis for a Class of Multistage Interconnection Networks," *IEEE Transactions on Computers*, vol. c-30, no. 10, 10/83.
- [17] Feng, Tse-yun. "A Survey of Interconnection Networks," *Computer*, vol. 14, no. 12, 12/83, 12-30.
- [18] Franklin, Mark A. "VLSI Performance Comparison of Banyan and Crossbar Communications Networks," *IEEE Transactions on Computers*, vol. C-30, no. 4, 4/81, 283-290.
- [19] Gilbert, E. N. and H.O. Pollak. "Steiner Minimal Trees." *SIAM J. Appl. Math.*, 16(1):1-29, 1968.
- [20] Giorcelli, S., C. Demichelis, G. Giandonato and R. Melen. "Experimenting with Fast Packet Switching Techniques in First Generation ISDN Environment," *International Switching Symposium*, 3/87.
- [21] Goke, L. R. and G. J. Lipovski, "Banyan Networks for Partitioning Multiprocessor Systems," *Proceedings of the 6th Annual Symposium on Computer Architecture*, 4/79, 182-187.
- [22] Gonet, P., P. Adam, J. P. Coudreuse. "Asynchronous Time-Division Switching: the Way to Flexible Broadband Communication Networks," *Proceedings of the International Zurich Seminar on Digital Communication*, 3/86, 141-148.
- [23] Haserodt, Kurt and Jonathan Turner. "An Architecture for Connection Management in a Broadcast Packet Network," Washington University Computer Science Department, WUCS-87-3.

- [24] Hayward, G., L. Linnell, D. Mahoney and L. Smoot. "A Broadband Local Access System Using Emerging Technology Components," *International Switching Symposium*, 3/87.
- [25] Hoberecht, William L. "A Layered Network Protocol for Packet Voice and Data Integration," *IEEE Journal on Selected Areas in Communications*, vol. SAC-1, no. 6, 12/83, 1006-1013.
- [26] Huang, Alan and Scott Knauer. "Starlite: a Wideband Digital Switch," *Proceedings of Globecom 84*, 12/84, 121-125.
- [27] Huang, Alan. "Distributed Prioritized Concentrator," U.S. Patent 4,472,801, 1984.
- [28] Huang, Alan and Scott Knauer. "Wideband Digital Switching Network," U.S. Patent 4,542,497, 1985.
- [29] Jaffe, J. M. "Distributed multi-destination routing: the constraints of local information." *SIAM J. Comput*, 14(4):875-88, November 85.
- [30] Jenq, Yih-Chyun. "Performance Analysis of a Packet Switch Based on a Single-Buffered Banyan Network," *IEEE Journal on Selected Areas in Communications*, vol. SAC-1, no. 6, 12/83, 1014-1021.
- [31] Karol, M. J., M. G. Hluchyj and S. P. Morgan. "Input vs. Output Queueing on a Space-Division Packet Switch," *Proceedings of Globecom*, 12/86.
- [32] Kermani, P. and L. Kleinrock. "Virtual Cut-Through: A New Computer Communication Switching Technique," *Computer Networks*, vol. 3, 1979, 267-286.
- [33] Khakoo, Shabbir and Jonathan Turner. "System Testing of a Broadcast Packet Switch," Washington University Computer Science Department, WUCS-87-4.
- [34] Kou, L., G. Markowsky, and L. Berman. "A fast algorithm for Steiner trees." *Acta Informatica*, 15:141-5, 81.
- [35] Kruskal, C. P. and M. Snir. "The Performance of Multistage Interconnection Networks for Multiprocessors," *IEEE Transactions on Computers*, vol. C-32, no. 12, 12/83, 1091-1098.
- [36] Kulzer, John J. and Warren A. Montgomery. "Statistical Switching Architectures for Future Services," *Proceedings of the International Switching Symposium*, 5/84.

- [37] Lea, Chin-tau. "The Load-Sharing Banyan Network," *IEEE Transactions on Computers*, 12/86.
- [38] Lee, Tony T. "Non-Blocking Copy Networks for Multicast Packet Switching," Bell Communications Research, 1987.
- [39] Melen, Riccardo and Jonathan S. Turner, "Distributed Protocols for Access Arbitration in Tree Structured Communication Channels," Washington University Computer Science Department, WUCS-87-17, 8/87.
- [40] Miller, R. E. and J. W. Thatcher. *Complexity of Computer Computations*, chapter "Reducibility Among Combinatorial Problems," by R.M. Karp, pages 85-103. Plenum Press, 1972.
- [41] Montgomery, Warren A. "Techniques for Packet Voice Synchronization," *IEEE Journal on Selected Areas in Communications*, vol. SAC-1, no. 6, 12/83, 1022-1028.
- [42] Muise, R. W., T. J. Schonfeld and G. H. Zimmerman III. "Experiments in Wideband Packet Technology," *Proceedings of the International Zurich Seminar on Digital Communication*, 3/86, 135-139.
- [43] Netravali, A. N. and J. D. Robbins. "Motion Compensated Television Coding: Part 1," *Bell System Technical Journal*, vol. 58, No. 3, 3/79.
- [44] Netravali, A. N. and J. O. Limb. "Picture Coding: A Review," *Proceedings of the IEEE*, vol. 68, No. 3, 3/80.
- [45] Patel, J. H. "Performance of Processor-Memory Interconnections for Multiprocessors," *IEEE Transactions on Computers*, vol. C-30, no. 10, 10/81, 771-780.
- [46] Pratt, W. K. "Image Transmission Techniques," Wiley (Interscience), New York, 1978.
- [47] Rayward-Smith, V. J. "The Computation of Nearly Minimal Steiner Trees in Graphs," *International Journal of Math. Ed. Sci. Tech.*, 14(1):15-23, 1983.
- [48] Rettberg, R., C. Wyman, D. Hunt, M. Hoffman, P. Carvey, B. Hyde, W. Clark and M. Kraley. "Development of a Voice Funnel System: Design Report," Bolt Beranek and Newman, Report No. 4098, 8/79.
- [49] Richards, Gaylord and Frank K. Hwang. "A Two Stage Rearrangeable Broadcast Switching Network," *IEEE Transactions on Communications*, 10/85.

- [50] Robbert, George. "Design of a Broadcast Translation Chip," Washington University Computer Science Department, WUCS-87-9.
- [51] Sincoskie, W. D. "Transparent Interconnection of Broadcast Networks," *Proceedings of the International Zurich Seminar on Digital Communication*, 3/86, 131-134.
- [52] Staehler, R. E., J. J. Mansell, E. Messerli, G. W. R. Luderer, A. K. Vaidya. "Wideband Packet Technology for Switching Systems," *International Switching Symposium*, 3/87.
- [53] Stroustrup, Bjarne, "The C++ Programming Language," Addison-Wesley, 1986.
- [54] Takeuchi, Takao, Hiroshi Suzuki, Shin-ichiro Hayano, Hiroki Niwa and Takehiko Yamaguchi. "An Experimental Synchronous Composite Packet Switching System," *Proceedings of the International Zurich Seminar on Digital Communication*, 3/86, 149-153.
- [55] Turner, Jonathan S. and Leonard F. Wyatt. "A Packet Network Architecture for Integrated Services," *Proceedings of Globecom 83*, 11/83, 45-50.
- [56] Turner, Jonathan S. "Fast Packet Switching System," United States Patent #4,494,230, 1/15/85.
- [57] Turner, Jonathan S. "Design of an Integrated Services Packet Network," *Proceedings of the Ninth ACM Data Communications Symposium*, 9/85, pages 124-133.
- [58] Turner, Jonathan S. and L. F. Wyatt, "Alternate Paths in a Self-Routing Packet Switching Network," United States Patent #4,550,397, 10/29/85.
- [59] Turner, Jonathan S. "Design of a Broadcast Packet Network," *Proceedings of Infocom*, 4/86.
- [60] Turner, Jonathan S. "New Directions in Communications," *IEEE Communications Magazine*, 10/86.
- [61] Turner, Jonathan S. "Design of an Integrated Services Packet Network," *IEEE Journal on Selected Areas in Communications*, 11/86.
- [62] Turner, Jonathan S. "Specification of Integrated Circuits for a Broadcast Packet Network," Washington University Computer Science Department, WUCS-87-5.

- [63] Turner, Jonathan S. "The Challenge of Multipoint Communication," Washington University Computer Science Department, WUCS-87-6. Also, to appear in *Proceedings of the ITC Seminar on Traffic Engineering for ISDN Design and Planning*, 5/87.
- [64] Turner, Jonathan S, "Fluid Flow Loading Analysis of Packet Switching Networks," Washington University Computer Science Department, WUCS-87-16, 7/87.
- [65] Verbeist, W. "Video Coding in an ATD Environment," *Picture Coding Symposium*, 6/87.
- [66] Waxman, Bernard. "Thesis Proposal: Routing of Multipoint Connections," Washington University Computer Science Department, WUCS-87-2.
- [67] Wirsching, J. E. and T. Kishi, "Conet: A Connection Network Model," *IEEE Transactions on Computers*, vol. C-30, 4/81.
- [68] Yeh, Y. S., M. G. Hluchyj and A. S. Acampora. "The Knockout Switch: a Simple Modular Architecture for High Performance Packet Switching," *International Switching Symposium*, 3/87.