ADVANCED COMMUNICATIONS SYSTEMS

Jonathan S. Turner (PI)
September 1, 1987 – August 31, 1988
WUCS-88-28

Mark A. Franklin
Guru Parulkar

Pierre Costa
Makoto Imase
Riccardo Melen

Akira Arutaki
Shahid Akhtar
Neil Barrett
Victor Griswold
Mark Hunter
Scott Johnson
Shabbir Khakoo
Tony Mazraani
Anne Reynolds
George Robbert
James Sterbenz
Einir Valdimarsson
Bernard Waxman

Research Objectives

The Advanced Communications Systems Project is concerned with new communication technologies that can support a wide range of different communication applications in the context of of large public networks. Communication 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 communication applications proliferate, it becomes clear that in the long term, a more flexible communication 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 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.

The ACS project began on January 1, 1986. Our current sponsors are

National Science Foundation (grant DCI 8600947)
Bell Communications Research
Bell Northern Research
Italtel SIT
Nippon Electric Corporation

We thank all our sponsors for their collaboration and support. Special thanks go to Gil Devey and Steve Wolff at NSF, Eric Nussbaum and Neil Haller of Bell Communications Research, Al Winterbauer and Dan Stevenson of BNR, 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 CS Department Chairman Jerry Cox for all their support and encouragement.

Contents

List of Figures

1. Summary of Progress

This report covers work performed from September 1987 through August 1988. This has been a productive year for us with substantial progress being made on a number of different fronts. In particular, we have developed new ideas on the design of practical switching systems that can support thousands of gigabit rate transmission links. We have made a significant generalization of the classical theory of nonblocking networks and have related it to the practical design of large scale packet switching systems. We continue to make good progress on our prototyping efforts with two new chip designs now complete and three others approaching completion; in support of this effort we have developed several design tools that represent significant contributions in their own right. We have studied a class of algorithms for video coding that yield substantially better compression than standard methods. And we have begun work on the problem of how to integrate high speed communication networks into a heterogeneous environment. Details of these and other efforts appear later in this report.

We continue to actively publish the results of our research. Papers have been presented at several conferences and revised versions have appeared or are scheduled to appear in leading journals; several theses have been completed; one patent has been awarded and an application for a patent on a hardware implementation of a buffer management system has been filed. (See Figures 1.1,1.2 for details.) Our work has generated a great deal of interest throughout the world, as evidenced by the many speaking invitations that have been received during the past year.

We summarize our progress in the following paragraphs. More detailed accounts of this work appear in later sections of the report.

Switch Architecture Studies

During the last year, we have been turning our attention to the architecture of packet switching systems that can support transmission link speeds in excess of a

Published Papers

"Fluid Flow Loading Analysis of Packet Switching Networks," by Jonathan Turner. *Proceedings of the International Teletraffic Congress*, June 1988. Also, submitted to *Computer Networks and ISDN Systems.*

"Distributed Protocols for Access Arbitration in Tree Structured Communication Channels," by Riccardo Melen and Jonathan Turner. *Proceedings of ICC 88*, June 1988. Also, submitted to *IEEE Transactions on Communications.*

"Design of a Broadcast Packet Switching Network," by Jonathan S. Turner, *IEEE Transactions on Communications*, June 1988.

"Broadcast Packet Switching Network," by Jonathan S. Turner, U.S. Patent #4,724,907, March 1988.

Invited Lectures

Bell Northern Research, Research Triangle Park, North Carolina (8/88)
Timeplex, Inc., Woodcliff Lake, NJ (8/88)
NEC, Tokyo, Japan (7/88)
NTT, Tokyo, Japan (7/88)
CSELT, Turin, Italy (5/88)
Italtel, Milan, Italy (5/88)
Digital Equipment Corporation, Littleton, MA (3/88)
University of California, Davis (2/88)

Tutorial on "Integrated Networks for Diverse Applications," at *Globecom 88* and UCLA Extension Short Course (2/88).

Program committee for *Computer Networking Symposium*, April 1988. Guest editor for special issue of *IEEE Journal on Selected Areas in Communications* on broadband packet communications

Filed patent application on buffer management system for multipoint packet networks (3/88).

Figure 1.1: Publications and Related Activities

---

gigabit per second and which scale economically to very large configurations (thousands or tens of thousands of transmission links). We have concentrated on systems using CMOS integrated circuit technology with very wide internal data paths using a bit-sliced structure. We have quantified the complexity advantages of bit-sliced structures, developed practical solutions to the associated control problems, and developed a fairly detailed paper design of a system that can support up to 32 thousand fiber optic links operating at speeds of 1.6 Gb/s using current CMOS technology with 100 Mb/s clock speeds and 32 bit wide data paths. It appears

"Design of Another VLSI Packet Switch Element," by Einir Valdimarsson, WUCS-88-23.

"Improved Search Algorithms for Video Codecs," by Shabbir Khakoo, Washington University Electrical Engineering Department, MS thesis, June 1988.

"A Circuit Generator for Synchronous Streams Processors," by George Robbert, Washington University Computer Science Department, MS thesis, May 1988.

"Worst-case Performance of Rayward-Smith's Steiner Tree Heuristic," by Makoto Imase and Bernard Waxman, WUCS-88-13.

"Towards a Framework for High Speed Communication in a Heterogeneous Networking Environment," by Guru Parulkar and Jonathan Turner, WUCS-88-7.

"Buffer Management System," by Jonathan Turner, WUCS-88-6.

"Probable Performance of Steiner Tree Algorithms," by Bernard Waxman, WUCS-88-4.

"Nonblocking Multirate Networks," by Riccardo Melen and Jonathan Turner, WUCS-88-2.

"Congestion Control in Fast Packet Networks," by Shahid Akhtar, Washington University Electrical Engineering Department, MS thesis, November 1987.

Figure 1.2: Theses and Technical Reports

---

likely that with near term technology improvements, this approach can be extended to handle speeds of 5–10 Gb/s. The details of this work appear in section 2.

Recently, we have also been exploring the problem of high speed switching systems for connectionless networks. While most researchers in the telecommunications community believe a connection-oriented approach is the most appropriate for high speed packet networks, there has not yet been a careful evaluation of the relative merits of connectionless and connection-oriented operation in a high speed networking environment. We are attempting to perform such an evaluation in order to give a clearer picture of the trade-offs involved. We have found that high speed connectionless networks are indeed possible, although they appear to be more costly to implement. The costs appear primarily in the packet processors that interface between the switch fabric and the transmission links. Connectionless networks require large buffers with priorities, link level and cross-switch flow control, plus content-addressable addressing tables. We have quantified the associated costs and estimate that a packet processor for a connectionless network might require about 20 chips to implement it (most being memory), as opposed to perhaps 4–5 for a connection-oriented network with comparable performance. While this is a clear advantage for connection-oriented networks, in some environments that cost may be less important than other factors.

Nonblocking Multirate Networks

In the past year, we have generalized the classical theory of nonblocking networks to cover switching systems in which a switch's internal data paths are shared among different connections. This theory is applicable both to multirate circuit switching systems and packet switching fabrics in which all packets within a connection follow the same path. We have derived conditions under which the Clos, Cantor and Beneš networks are strictly or rearrangeably nonblocking. In particular, we have shown that for multirate traffic, an i stage Beneš network is strictly nonblocking if the internal data paths are i times faster than the external transmission links. This is an important practical result since with 32 port switch elements we can construct 1024 port switching fabrics in three stages. Given a 3:1 speed advantage, such a network becomes strictly nonblocking, which is potentially important in an environment where the distribution of connection bandwidths may vary widely. Even for smaller speed advantages, we expect excellent blocking behavior and are now beginning to study the blocking characteristics of such switches.

Recently, we have extended our work on nonblocking networks to networks supporting multipoint communication. We have shown that classical results due to Pippenger and Thompson can be extended to the multirate environment. We have also shown that a pair of i stage Beneš networks placed back-to-back forms a wide-sense nonblocking network for multipoint connections if we have a speed advantage of i. This configuration, together with the routing algorithm used to obtain nonblocking operation are both novel and we plan to recommend an appropriate patent filing.

Prototype Switch Design

Work on a laboratory prototype of our switching system has been progressing well. Four integrated circuits implementing preliminary versions of the packet switch element and broadcast translation circuits have been received back from fabrication and have been tested. While we have had mixed results with these chips, we have learned a great deal from this process and have incorporated the lessons learned in the current versions, which have eight bit wide data paths and can support substantially higher clock speeds. The eight bit version of the packet switch element was submitted for fabrication at the end of July and is expected back from fabrication at the end of September. In addition to the wider data paths, this design incorporates architectural modifications in order to achieve higher clock speeds. We have also completed and fully simulated the design of a general purpose packet buffer, which will be used within the packet processor circuit. This chip will be submitted for fabrication in the near future. Three other chips are now in the

process of being designed. Two chips implementing parts of the packet processor and one implementing the broadcast translation chip are now in fabrication. These are being designed with the aid of a circuit generator written for that purpose. This is making the design of these chips much more rapid and less error prone than it would otherwise be.

Design Tools

In support of our prototyping efforts we have developed several supporting design tools. The most ambitious is a circuit generation program that can be used to quickly layout a large class of common circuits required within the packet processor, broadcast translation circuit and other common subsystems. This program allows the user to specify a component of a system in a functional notation similar to a conventional programming language. It then translates this specification into a circuit satisfying the specification. This program has been written by George Robbert as part of his masters thesis research [80] and is now being applied to design of several of the major components within the packet processor and broadcast translation circuit.

We have also developed tools for generating packet buffers and lookup tables of different sizes and configurations. These tools will also allow us to more rapidly implement different components of our prototype system. Another tool that we are using extensively is a program to generate control and timing circuits from a high level specification. The circuits generate timing signals that can be qualified by a set of control inputs. This program was written originally for use within the circuit generator mentioned above but has been proven to be more generally useful and is now being used within the packet switch element and packet buffer chips. One of the more time-consuming parts of designing integrated circuits is generating the test vectors needed for logic simulation. We have developed an approach to generating these test vectors that allows us to generate both the input test vectors and expected output vectors using a special-purpose test vector generation program. We have developed such programs for both the packet switch element and packet buffers and have found them extremely helpful, allowing us to more thoroughly test these designs than we could have by manual methods.

Connection Management

Connection management refers to the collection of algorithms used to create and maintain multipoint connections in a broadcast packet network. A multipoint connection is intended to be a flexible mechanism that can support a wide variety of different applications. In our last progress report, we described our approach

for specifying general multipoint connections, an architecture for a connection management software system and a set of protocols to implement that architecture.

We have completed development of an initial implementation of this connection management system in the form of a software simulation that allows us to configure an arbitrary network, then set up and modify multipoint connections in that network. Our implementation of multipoint connections includes a general transaction mechanism for sequencing concurrent changes to a connection. The software was implemented first on a VAX 11/750 and has since been ported to a Sun workstation environment; in this new context, we are developing a graphical user interface to allow simpler specification of network configurations as well as better observation and control of connections in progress. We expect this graphical interface to form the basis of some graphical network management tools that we hope to develop in the coming year. The simulation has proved very useful in testing out our ideas on multipoint connection management protocols. Based on experience obtained to date, we are now refining these protocols to make them simpler and more consistent at both the network access level and the internal network level. In the coming year, we will implement these refinements and add the lower level software required to control the protoype system under development.

Multipoint 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. Networks supporting multipoint communication channels of arbitrary bandwidth raise a variety of new issues for routing algorithms. We have primarily studied the formulation of the routing problem in which we seek to identify a shortest subtree within the network that contains the endpoints to be joined by a given connection and has sufficient bandwidth for the connection [102]. This formulation leads to a Steiner tree problem, which is known to be NP-complete.

We have experimentally evaluated several approximation algorithms for the Steiner tree problem. In the previous report, we described results for the so-called *minimum-spanning tree heuristic* (MST) and a dynamic greedy algorithm. In the last year, we have also evaluated an algorithm proposed by Rayward-Smith; we have recently completed an analysis of this algorithm's worst-case performance and shown that it is no better than that of MST [42]. On the other hand, our analysis has suggested a variation on Rayward-Smith that we expect may have better worst-case performance; also, its average case performance (based on exerimental evaluation) is somewhat better than MST, although both are very good. We have also evaluated a weighted version of the greedy algorithm for the dynamic version of the routing problem and have found that for appropriate choice of the weights,

this algorithm gives better average performance than the simple greedy algorithm; more significantly, the weighted algorithm is less subject to pathological behavior than the simple one.

We have recently begun work on a simulation environment that will permit us to evaluate our algorithms under more realistic conditions. In particular, this environment will allow us to simulate a network involving multiple connections and in which the routing algorithms are fully distributed. We expect to have performance results based on these simulations in time for the spring progress review.

Packet Video

Packetized transport of video signals raises a variety of important issues that we have been exploring. 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. In the last year, we have studied hybrid coding algorithms employing transform coding, motion compensation and adaptive quantization. We have discovered that the commonly used search algorithms for motion compensation perform poorly in

the presence of moderate to high motion. While they work adequately in video conferencing situations (which typically involve very little motion), they do poorly in more general contexts. We have developed a new class of *signature-based* search algorithms, which compute a concise signature for each position in the search space and match the current sub-block against each signature. We have evaluated one set of algorithms in this class and have found it increases the effective compression by a factor of three or four during rapid motion [50].

High Speed Internetworking

In our earlier work, we have concentrated on high speed networking in the context of a homogeneous environment. This is also typical of the approach taken by other groups working on high speed communication systems, but is in some ways unrealistic as it fails to explicitly take into account the diversity of existing and future networks, and the resulting need for inter-operation among separately administered and/or technologically dissimilar networks. In the last six months we have begun work on a framework for allowing diverse networks to inter-operate, while supporting both very high speed applications and multipoint communication [68]. This framework follows the general approach to interworking adopted in the ARPA internet protocols, but extends it in several respects. First it adds a connection-oriented transport service at the internet level, that can support applications with demanding performance requirements. Second it includes a more general addressing scheme, to allow interworking among diverse subnets. Third, it provides a framework for parametric description of subnet capabilities and connection requirements, allowing the routing of connections through subnets with appropriate capabilities in an application-independent fashion.

A connection-oriented transport service is important for several reasons. Perhaps the most obvious is performance. Connection-oriented systems separate the more complex control operations from data transfer, allowing simple and fast hardware implementations of the data transfer. Connection-oriented networks are also attractive because they allow the network to make explicit resource allocation decisions when connections are established, and this in turn makes it possible to offer far more predictable performance than in connectionless networks. Finally, connection-oriented networks offer more generally useful methods of multipoint communication than are possible in truly connectionless networks.

In our work we envision interoperation among a much wider class of networks than envisioned by the current internet model. In particular, we would like to support inter-operation between high speed packet networks, the current ARPA internet, X.25 networks and the public telephone network. Addressing is a key issue in allowing this level of diversity. We have proposed an addressing scheme that

would accommodate such diversity without requiring that the individual subnetworks abandon their native addressing mechanisms.

Given the variety of capabilities of the subnetworks included in an extended internet, we feel it is essential that the internet protocol include mechanisms for describing the capabilities of subnetworks, so that routing decisions can be guided by this information. For example, when selecting a route for a connection requiring a bandwidth of 1 Mb/s it is essential that the route not traverse subnetworks incapable of supporting that bandwidth. Similarly connections requiring low packet loss rates should not be routed through subnets that lose packets frequently.

Administrivia

In the past year, our research team has expanded with the addition of two new faculty members. We now have three faculty members involved in the project, one full-time staff person, one visiting research associate, ten graduate students and two part-time undergraduate students. Guru Parulkar joined the Computer Science Department in September 1987, after completing his PhD at the University of Delaware. Dr. Parulkar's thesis research focussed on the design and analysis of highly reliable local area networks based on flooding protocols. He is now concentrating on the problem internetworking of high speed packet networks. Andreas Bovopoulos is our most recent new faculty member, who has joined us from Columbia University where he studied the performance of flow control and routing protocols in networks. Riccardo Melen who was here as a visiting researcher in 1987 has returned to CSELT. His year here was a very productive one; he co-authored two papers with Dr. Turner [59,60] and co-invented a novel multipoint switching fabric. In March, Dr. Makoto Imase of NTT joined us for a one year visit. He has begun working with Buddy Waxman on the multipoint routing problem and their collaboration has already resulted in the solution of an important open problem in this area; a paper describing this work has been submitted for publication.

Among the students participating in the project, several changes have taken place. Four students have completed masters degrees. Shahid Akhtar graduated last fall and is now working at Bell Northern Research, in Research Triangle Park, NC, George Robbert graduated in May and is now with Hewlett-Packard in Fort Collins, CO, Shabbir Khakoo graduated in June and is working for AT&T Information Systems in Middletown, NJ and Mark Hunter who also graduated in June is working for McDonnell Douglas in St. Louis. All four of these students made strong contributions to the project and we wish them success in their future endeavors. We have several students who have joined the project in the past year. Tony Mazraani joined the project in January and has been working initially on several projects related to our prototype development effort. Gaurav Garg and

Name	Degree (exp. grad.)	Research Area
Neil Barrett	MS (5/89)	memory use in fast packet switching systems
Einir Valdimarsson	MS (5/89)	blocking probability in fast packet networks
Bernard Waxman	DSc (5/89)	multipoint routing
Victor Griswold	DSc (12/89)	distributed program debugging
Tony Mazraani	MS (12/89)	communication circuit design
James Sterbenz	DSc (12/89)	interfacing computers to high speed networks
Akira Arutaki	DSc (5/90)	switching architectures
Haifeng Bi	MS (5/90)	communication circuit design
Gaurav Garg	MS (5/90)	communication circuit design
Michael Gaddis	DSc (5/91)	connection management

Figure 1.3: Current Graduate Students

Name	Degree	Date	Dept.	Thesis Title
Shabbir Khakoo	MS	8/88	EE	Improved Search Algorithms for Video Codecs
Mark Hunter	MS	8/88	CS	
George Robbert	MS	5/88	CS	A Circuit Generator for Synchronous Streams Processors
Shahid Akhtar	MS	1/88	EE	Congestion Control in Fast Packet Networks
Rick Bubenik	MS	8/85	CS	Performance of a Broadcast Packet Switch

Figure 1.4: Graduates

Haifeng Bi joined the project this summer and have also been involved in the prototype efforts. Mike Gaddis has just joined the project this fall and will probably be working in the area of connection management and internetwork protocols. We also have two part-time undergraduate students working on the project; Scott Johnson started last fall and has designed several graphics programs and has been assisting with some of the prototype design. Anne Reynolds started this summer and has been working on integrated circuit testing, among other things.

Our funding picture is fairly healthy. Bell Northern Research joined the project

in earlier this year, becoming our fourth industrial sponsor. Our support from the National Science Foundation has remained stable, but we will have to apply for new NSF funding this fall, as our current grant expires in June of next year. 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.

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, one technical staff person, plus seven graduate students, on the third floor of Bryan Hall, across from our main laboratory facility. This laboratory houses most of our computers, and a cluster of terminals and workstations for graduate student use and also serves as an informal meeting room. We also have additional office and laboratory space on the fifth floor of Bryan. Seven students and two additional staff members are located in this area. While our space situation is adequate, it is somewhat cramped and is likely to remain so for the next 18 months, while construction of a new 50,000 square foot research building is underway. When that building is completed, it will mean substantial new space for the Computer Science and Electrical Engineering Departments. We expect that at that point, we will move into the fourth floor of Bryan Hall, consolidating our group in one area and giving us additional space to work with.

The Center's base of equipment includes a VAX 750, a MicroVax II/GPX and a Sun workstation environment including a 3/280 file server, a 3/150 which will interface to our prototype switching system, and six 3/50 diskless workstations. In the last year a Sun 3/60 and a Sun 3/160 workstation have been added to that configuration. The 3/60 has supported our integrated circuit design efforts and the 3/160 will be used as the connection processor for our prototype switch. We have also doubled the disk storage on our file server to over one gigabyte, in response to the demands imposed by our VLSI design activities. The Center has also acquired a 64 processor NCUBE parallel computer, which Professor Franklin will be using to support his research in the area of design automation. We also anticipate its possible use in our project. In addition we have about fifteen conventional terminals, a PC/AT, another VLSI design station, several printers, and 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 construction of the new building should provide ample space in the longer term, there will be an intermediate period of limited space that will have

to be managed carefully.

2. Switch Architecture Studies

Faculty	Jonathan Turner
Graduate Students	Akira Arutaki
	Neil Barrett
	Haifeng Bi
	Einir Valdimarsson

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 [93], we continue to evaluate alternatives, in order to identify possible improvements. In the past year, we have begun an examination of systems that can support gigabit rate transmission using a bit-sliced internal architecture. Such systems represent a substantial improvement in performance and economy of implementation over current designs. We have also begun studies of global memory organization in large switching systems, queueing and blocking behavior in such systems and a study of switch design options for connectionless packet networks. Finally, we have extended an earlier simulation program used to assess our current design, by providing a graphical display that shows the status of the simulated network and provides interactive control of traffic sources and monitoring.

2.1. Bit-Sliced Switch Fabrics

Packet switching fabrics employing parallel data paths can be organized in a couple different ways. One possibility is the so-called *word-serial* approach, in which all the bits in a given data path pass through the same physical components. Another is the *bit-sliced* approach, in which the components making up the switch fabric are "sliced" so that each bit of the data path passes through a different set of components. An example illustrating these two approaches is shown in Figure 2.1. In the figure on the left, each box corresponds to a single integrated circuit, as do the rectangles on the right. Notice that each of these structures implements a 16

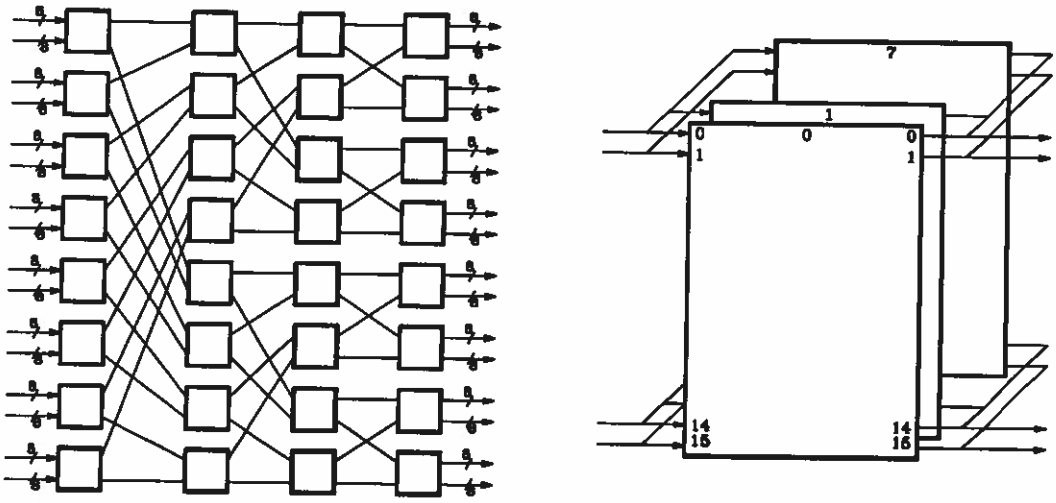

Figure 2.1: Comparison of Word-Serial and Bit-Sliced Organizations

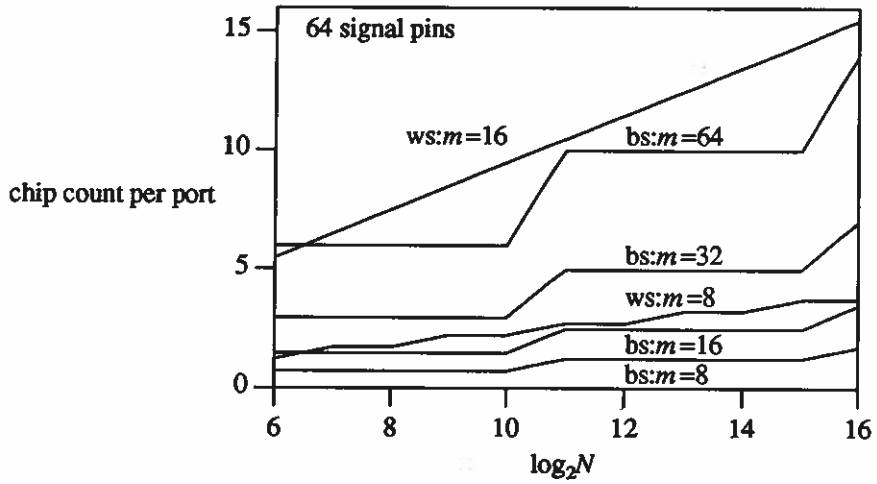

Figure 2.2: Complexity of Word-Serial and Bit-Sliced Beneš Networks

port switching fabric with 8 bit wide data paths and that the integrated circuits in both cases require 32 signal leads. However, the word-serial structure requires 32 chips while the bit-sliced structure requires just eight. For large systems, this advantage of bit-sliced structures becomes even more dramatic. Figure 2.2 plots the chip count per port for Beneš networks with several choices of the data path width. N is the number of ports the switch fabric has, m is the data path width, *ws* stands for word-serial and *bs* for bit-sliced. Notice that for the bit-sliced organization, we can achieve data path widths of 32 at a cost of about five chips per port for switches with between 2,048 and 32,768 ports.

Of course, on the other side, the word-serial organization is somewhat simpler

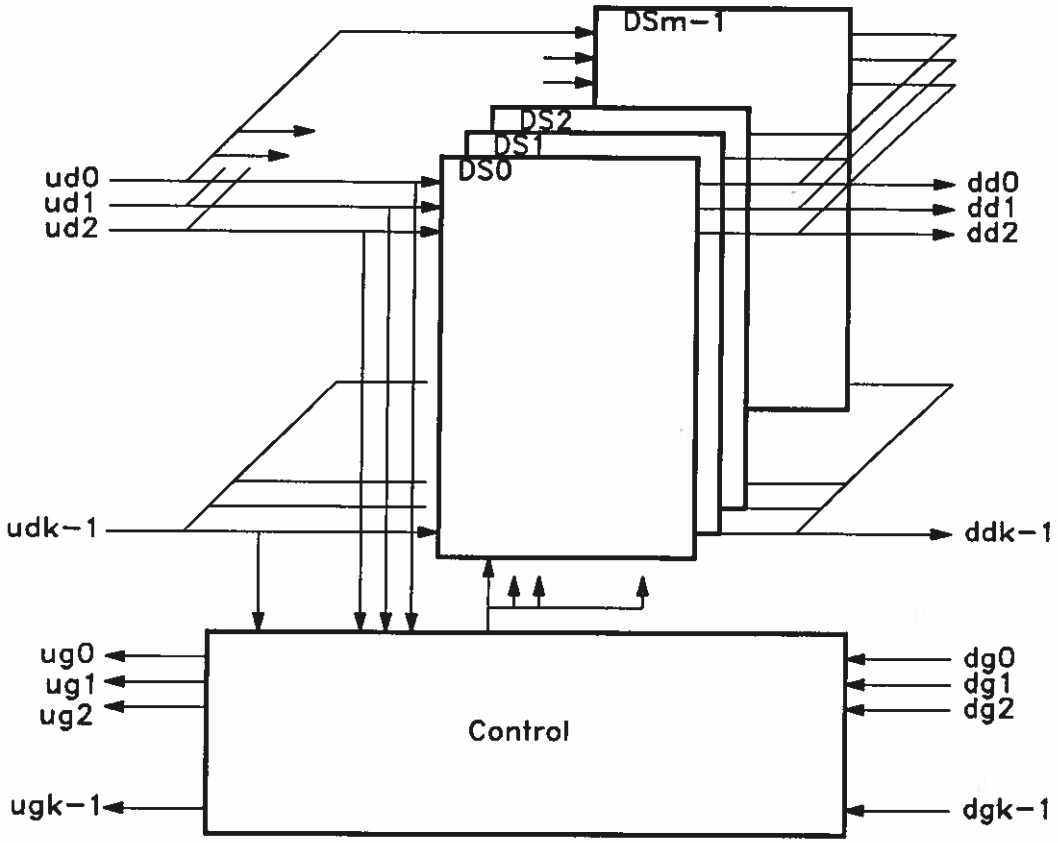

Figure 2.3: Bit-Sliced Switch Element

to control. In the bit-sliced organization, each slice must make the same control decisions for the packets it receives. This can be done either by replicating the control information and sending it to each slice, or by having one slice decode the control information, make the appropriate decisions and communicate the results to the other slices. We examine the latter alternative and describe a practical design of a packet switching element that implements it.

Figure 2.3 illustrates a design of a bit-sliced switch element with k input and output ports and supporting m bit wide data paths. Typical values for k might be 16 or 32. Values for m might range from 8 to 64. Packets enter on one of the k *upstream data* lines at left, and the m bits of each packet are distributed across m separate *data slices* (DS). The packets exit from the switch element on the *downstream data* lines at the right. The switch element contains sufficient internal buffering to store several packets for each port and implements a simple hardware flow control mechanism to prevent packets from overflowing these buffers. We will describe three versions of the data slice; one with the buffering on the input side, another with the buffering on the output side and a third with shared buffering.

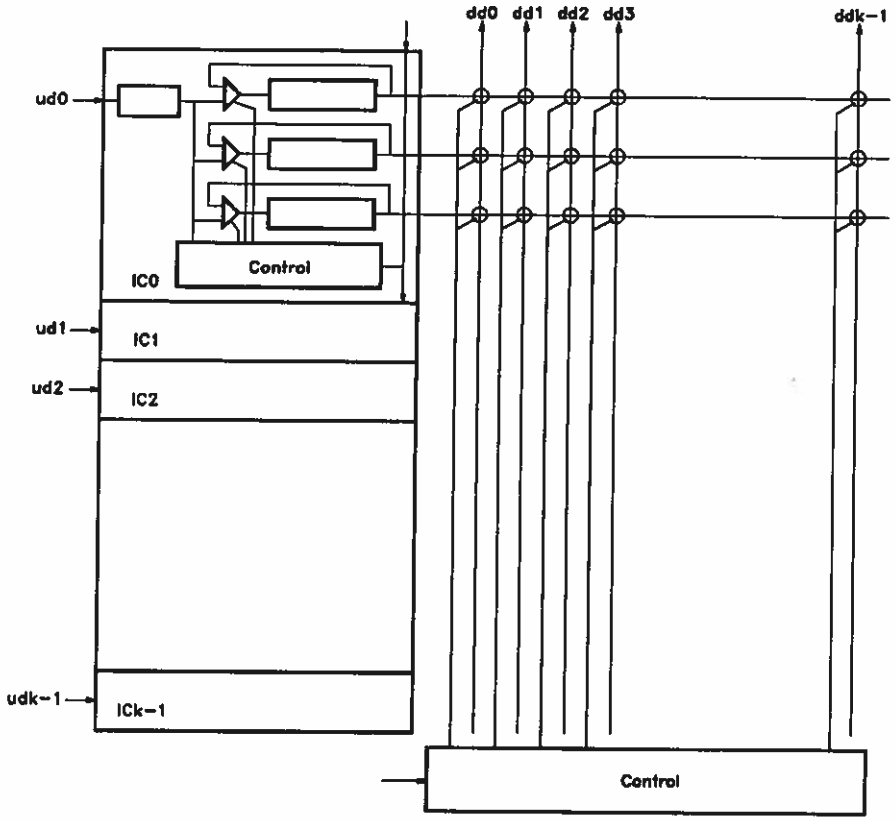

Figure 2.4: Data Slice with Input Buffering

The *control slice* shown at the bottom of the figure contains the circuitry used to control the operation of the switch element. It receives a set of *downstream grant* signals from the downstream neighbors and generates a corresponding set of *upstream grant* signals which are sent to the upstream neighbors. In general, a switch element asserts an upstream grant signal ug_i if it is prepared to receive a packet on the upstream data lines ud_i. The packets flowing through the switch element are organized so that all the control information (in particular, the addressing information) passes through the first data slice DS_0. This allows the control circuit to easily monitor the control information for all packets entering the data slice. Using this information, together with the downstream grants and the internal status of the switch elements, it makes control decisions and broadcasts those decisions to the data slices. In addition, the first bit of the packet in *every data slice* is a control bit indicating the presence or absence of a packet.

Figure 2.4 shows the organization of a chip implementing a data slice in which

the buffering is placed on the input side. The chip contains one *Input Circuit* (IC) for each input. Each IC contains several buffers, each able to store a complete bit slice of a packet. The buffers can be implemented as dynamic shift registers with a feedback path used to recirculate a packet if it is unable to proceed during a given cycle. The control circuit within the IC, keeps track of which buffers contain packets, detects the arrival of new packets and steers them to the first empty buffer. During one operational cycle, the control slice sends each IC one bit of control information for each buffer slot that the IC controls. If that bit is high, any stored packet in the buffer is recirculated, if it is low, any stored packet is transmitted from the buffer and the buffer becomes available to receive a new packet. The data slice also contains a crossbar matrix which provides access to the downstream data lines. During any given cycle, one crosspoint from each column of the crossbar is closed. The selected crosspoint is determined by a control register whose contents is determined by the control slice. Notice that the crossbar organization allows a packet to be sent to multiple outputs during a given cycle, permitting multipoint connections. Also notice that a packet that must be sent to several outputs need not be sent to all outputs simultaneously. If not all outputs are immediately available, it can be sent to the available outputs and recirculated in the input buffer until the remaining outputs become available.

The control slice is shown in Figure 2.5. This chip does not include any data storage; it merely monitors the bit slice containing the control information, makes the appropriate decisions and transmits these decisions to each of the data slices. The chip has an *Input Control Circuit* (ICC) for each of the k inputs. It also has a set of k *downstream grants* and k *upstream grants.* A downstream grant is asserted by one of a switch element's downstream neighbors if the neighbor is able to receive another packet. Similarly, the switch element asserts its upstream grant for each input that is able to receive another packet. The bit slice containing control information enters the chip on the *upstream data leads.* Each ICC shifts in the control information, latches it and decodes it. It is then stored in one of several control registers corresponding to the data buffers in which the packet data is stored. The control registers (CTL) independently contend for access to one or more outputs by sending their requests to an arbitration circuit.

The arbitration circuit consists of an array of *arbitration elements* (AE) together with an *arbitration tree* (AT) for each output. During a given operation cycle, a control register transmits to one row of arbitration elements, the range of outputs it requires access to; for example, it might request outputs 5–14. Each arbitration element in the row decodes the requested range and determines if its associated output has been selected. If so, it contends for the output by sending a request to the arbitration tree. Each arbitration tree is structured as a simple binary tree which accepts requests from the arbitration elements in one column of the

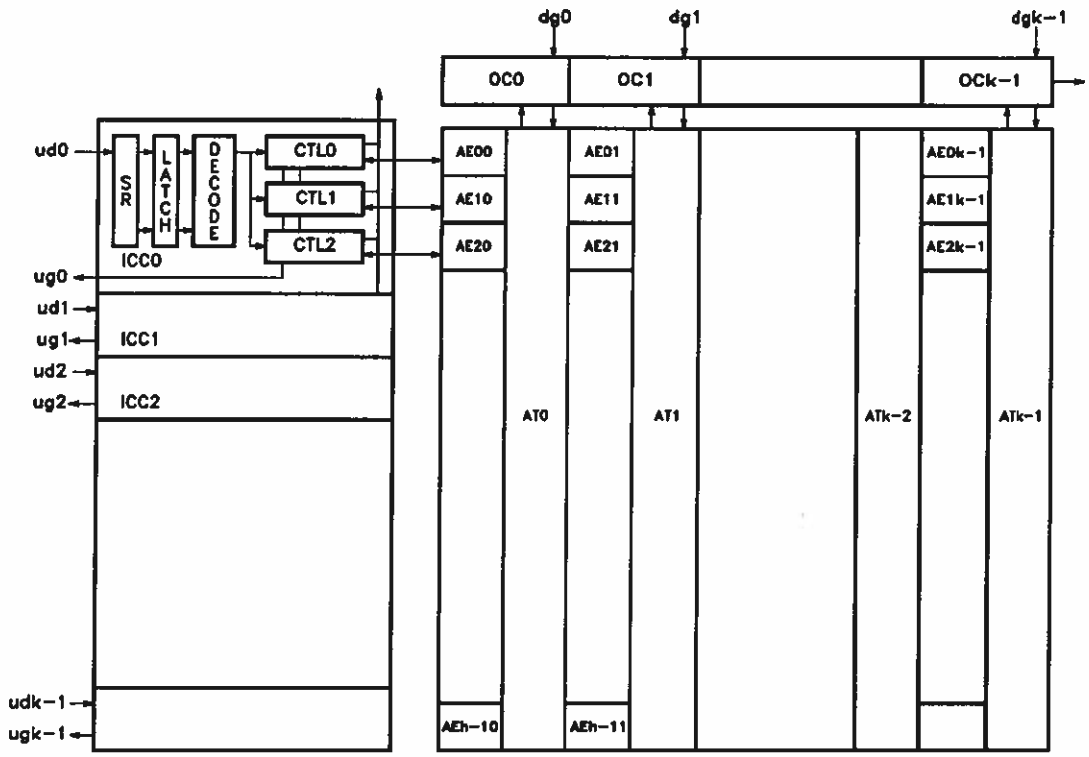

Figure 2.5: Control Slice

arbitration circuit and selects one of the contending AEs. If the downstream grant for that output indicates that the downstream neighbor is ready to receive a packet, the winner of the contention receives an acknowledgement indicating that it can transmit its packet. The *output control circuits* (OCC) at the root of the arbitration trees send the identities of the winning contenders to the data slices, where they are used to control the output crossbar. Each arbitration tree in fact makes its decision based on a priority sent to it from the arbitration elements. The high order field of the priority is the number of operation cycles that the contending packet has been waiting; this ensures that packets following the same path through the switch are served in first-in-first-out order and that no packet is kept waiting too long. It's of course also possible to implement more sophisticated priority schemes using the same basic mechanism. If a packet is to be transmitted to multiple outputs but does not receive access to all the required outputs on a particular cycle, the arbitration elements inform the control registers of that fact, and re-contend on the next cycle, continuing in this fashion until the packet has been transmitted to all required outputs. When the control circuit is informed that the packet has been transmitted to all outputs, it becomes available to receive a new packet.

Let $C_d(k, m)$ be the complexity of a data slice with k inputs and designed for

an m bit wide network, and let $C_c(k)$ be the complexity of the control slice. Then,

$$C_d(k,m) \approx kLBx_1/m + k^2x_2 \qquad \text{(data slice)}$$
$$C_c(k,m) \approx kx_3 + k^2Bx_4 \qquad \text{(control slice)}$$

where L is the total packet length in bits, B is the number of buffer slots per port, x_1 is the cost per bit of memory, x_2 is the cost per crosspoint, x_3 is the cost for one IC plus one OC and x_4 is the cost for one AE plus one AT node. Based on preliminary designs, we estimate x_1 at eight transistors, x_2 at $8+\log_2(1+Bk)$ transistors, x_3 at 1000 transistors and x_4 at 200 transistors. If we select $L = 4096$, $B = 3$ we have for example, $C_d(32,32) \approx 114,000$ transistors and $C_c(32,32) \approx 646,000$ transistors. These estimates show that while the data slice is clearly feasible using 2 micron CMOS technology, the control slice is at best marginal for 2 micron technology. It could however be implemented in a 1 micron technology with no problem.

Another important parameter of the design is the amount of control information that must be sent from the control slice to the data slices during each packet cycle. If we let $I(k,B)$ denote the amount of information required for a switch element with k ports and B buffers per port then

$$I(k,B) = kB + k\lceil\log_2(1+kB)\rceil$$

The time needed to transmit the control information to the data slices puts a lower bound on the length of a packet cycle. If r pins are used to carry the control information, then the packet length should be at least $mI(k,B)/r$. For example, if $k = m = 32$, $B = 3$ and $r = 8$, we require a packet with at least 1280 bits. A more realistic packet length is probably about twice this. For 64 bit wide data paths, the required packet length would of course double.

Figure 2.6 shows an alternative design for a data slice in which most of the buffering is placed on the output side. A single slot buffer is required for each input to prevent packets from being lost. The control information for the ICs, OCs and crossbars is received from the control slice. The circuit complexity of the data slice is approximately equal to that for the input buffering case. The control slice is perhaps a little more complicated as it requires a more sophisticated output arbitration tree. The amount of information required to control the data slice is given by

$$2k + k(B-1)\lceil\log_2(k+1)\rceil$$

In most cases, this is substantially higher than that required for the input buffered data sliceed data slice.

Figure 2.7 shows a third design in which the buffering is shared equally among the inputs and outputs. In this design, the packets enter at the top left, pass

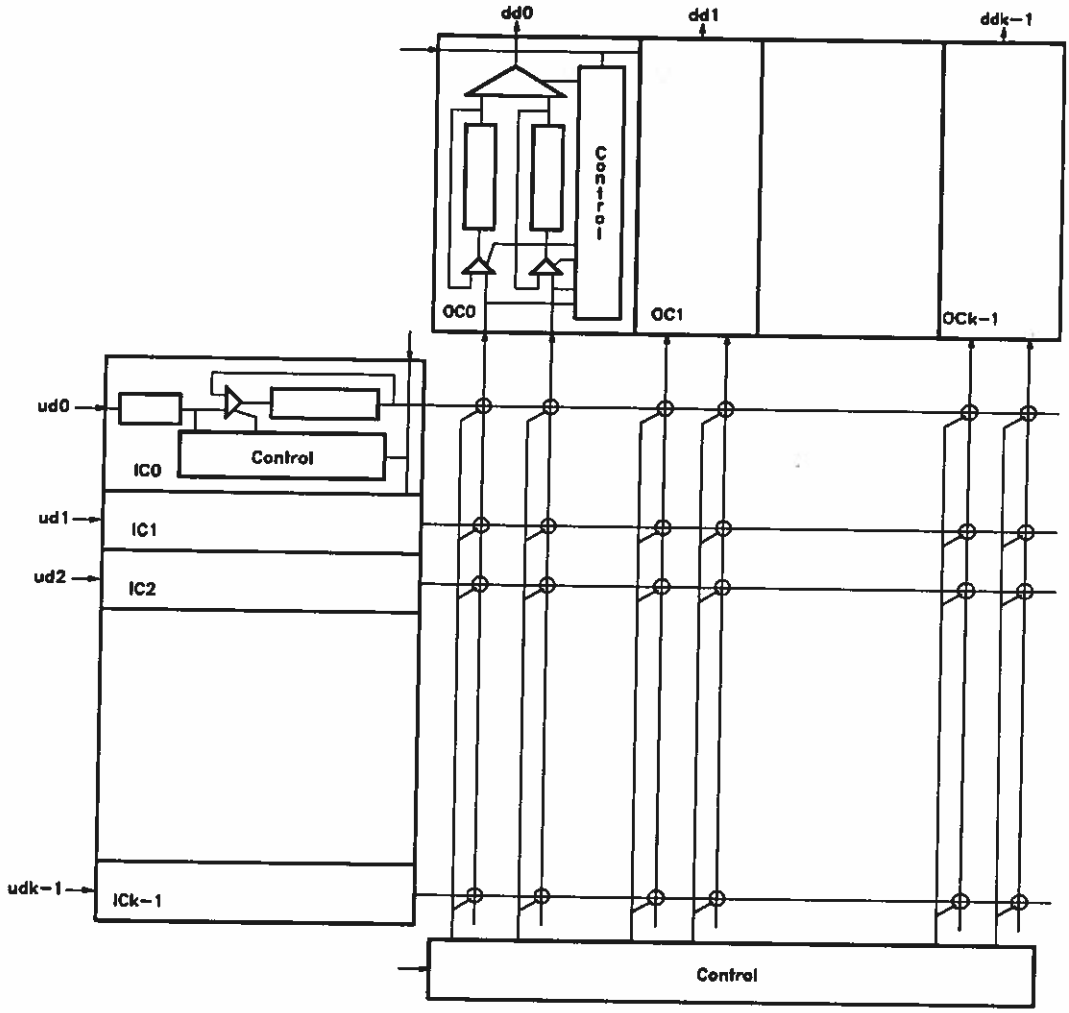

Figure 2.6: Data Slice with Output Buffering

through an input crossbar to a set of buffers, before being sent through an output crossbar to the downstream data lines. The buffers and output crossbar are controlled in exactly the same way as for input buffering. The input crossbar requires no external control information. In this design, the complexity of the data slice is approximately

$$kLBx_1/m + k^2(x_2 + x_5)$$

where x_5 is the complexity of an input crosspoint. If we estimate this at 50 transistors, the complexity of a switch element with $k = m = 32$, $L = 4096$ and $B = 3$ is approximately 165,000 transistors. The control slice is almost identical to the input buffering case, as is the amount of information required to control the data slice.

Based on this design we have estimated that a packet switching system com-

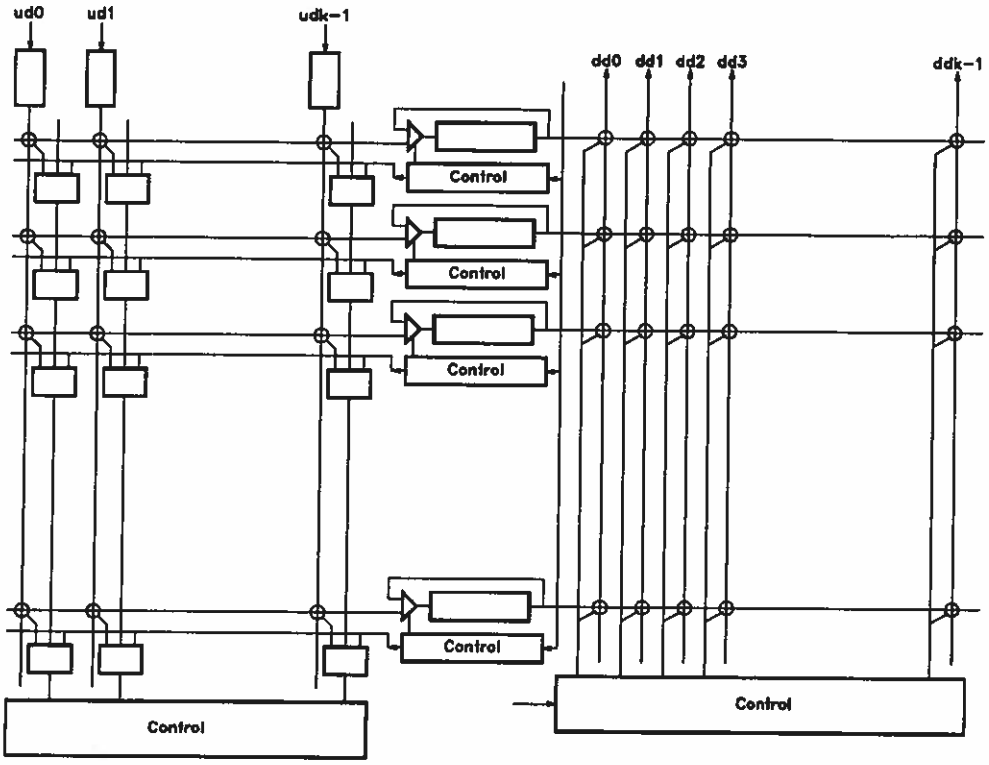

Figure 2.7: Data Slice with Shared Buffering

prising a Beneš network with 32 bit wide data paths and supporting 4096 fiber optic data links would require about twelve standard equipment cabinets. Another four cabinets would be required for the link interfaces and packet processors. If such a system was operated with a clock rate of 100 Mb/s (an achievable rate even with CMOS), its internal data paths would operate at an effective rate of 3.2 Gb/s, which is sufficient to support external link speeds of 1.6 Gb/s. This represents an order of magnitude improvement over the speeds being achieved by current research prototypes. While there are limits to how far such techniques can go, it appears likely that at least another factor of four is possible through a combination of higher clock rates and greater parallelism.

2.2. Switches for Connectionless Networks

Most work on high speed packet switching has centered on systems that transfer data using logical connections rather than datagrams. There are several good reasons to prefer connection-oriented networks. First and most important, in a

connection-oriented network it is possible to allocate resources to individual applications, allowing the network to provide a predictable performance level to an application even when the network is heavily loaded. Because the network has knowledge of the resource requirements of user applications, it can block new traffic when it lacks the resources to support it. In a connectionless network, when the network becomes overloaded all users experience degraded performance. A second advantage of connection-oriented networks is that they are simpler to implement, requiring only the simplest routing algorithms, no link level flow control and relatively modest buffer requirements. Connection-oriented networks also permit the implementation of general multipoint communication, something that is possible in connectionless networks only by introducing the idea of a multicast address, which is in fact nothing more or less than a connection by another name.

On the other hand, connectionless networks have their attractive points as well. First, because there is no need to establish a connection, the use of connectionless networks is much simpler. In particular, for many applications, determination of resource requirements in advance is difficult or impossible. Since connectionless networks don't require such a determination they are easier to use. Also, the design of network equipment is simplified somewhat by the elimination of connection management. Second, connectionless networks are inherently more robust than connection-oriented networks; because they lack the state information that defines connections, they can recover from failures of switches and transmission links without perceptible impact on users. Third, connectionless networks can distribute traffic more evenly, since packets are individually routed; of course this has the drawback that packets can be delivered in the wrong order.

In this section, we examine the possibility of implementing switching systems for high speed connectionless networks. Our objective is to get an understanding of the complexity of such systems so that we can make comparisons between them and systems for connection-oriented networks. We conclude that while high speed switching is connectionless networks is more complex in some respects and offers less predictable performance when heavily loaded, it is possible to build systems that match the raw speed of connection-oriented networks. Such systems may merit consideration in environments where the ease of use and inherent reliability of connectionless networks offset their drawbacks.

We start with the issue of buffering and flow control in connectionless networks. Because the flow of traffic in connectionless networks is unpredictable, flow control appears to be required to prevent excessive loss of packets during local overloads and to help steer traffic around congested areas. Flow control may be required both across links and between the input and output ports of switching systems.

Flow control across transmission links can be readily implemented in hardware so long as it is not coupled to error recovery as in conventional window-based

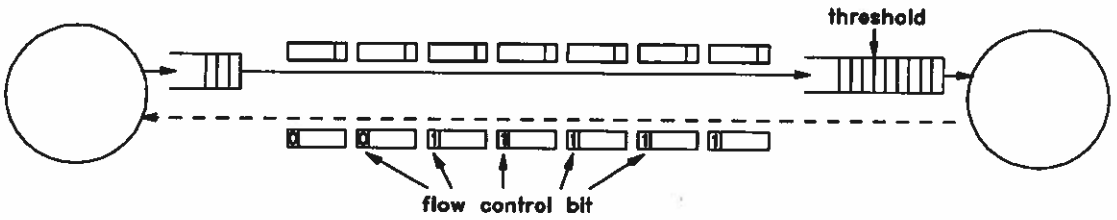

Figure 2.8: Simple Flow Control Across a Link

protocols. Figure 2.8 illustrates the basic idea. At the receiving end of the link is a buffer whose level is monitored at all times. When the level is above a threshold, a flow control bit is set in all packets sent in the reverse direction. When the level is below the threshold level, the flow control bit is cleared. When there are no user packets to send in the reverse direction, dummy packets are sent with the flow control information. In order to prevent buffer overflow and realize maximum link throughput, the receiving buffer should have a storage capacity that is four times that of the transmission link and the threshold should be set at the half full level. The storage capacity of a transmission link is given by RL/c where R is the rate bits are transmitted on the link, L is the length of the link and c is the propagation speed of the signal, which is approximately 200,000 km/s for optical fiber. So for example, a 1000 km link operating at a speed of 100 Mb/s has a bit capacity of 0.5 Mb, requiring a buffer size of 2 Mb. This goes to 20 Mb for a link operating at 1 Gb/s. Even a 20 Mb buffer could be reasonably constructed using video RAMS, or similarly organized memories. Commercial video RAM chips are available in sizes up to 1 Mb, so the 20 Mb buffer would require 20 memory chips meaning that a packet processor containing such a buffer would probably consume a large fraction of a circuit board and cost one to two thousand dollars. While this is higher than comparable costs in connection-oriented networks, it may not be unreasonable in certain environments. Also note that for short links (say 10 km), much smaller buffers could be used, allowing inexpensive packet processors to be used in access links where the cost sensitivity is highest.

Figure 2.9 illustrates how flow control can be performed across a switch using a control ring that contains one bit of flow control information for each outgoing link. In this example, the switching fabric is assumed to be a buffered Beneš network with flow control between switch elements. The outgoing packet processors (PP) have buffers capable of holding say, 20 packets and when the buffer occupancy exceeds a threshold, the PP sets the flow control bit. This prevents the input PPs from sending more packets to the overloaded PP, although packets within the switch fabric will still reach it. The use of flow control within the switch ensures that packets are not lost. The flow control ring limits the size of the switch somewhat. Note however that the ring need not make a full circulation on

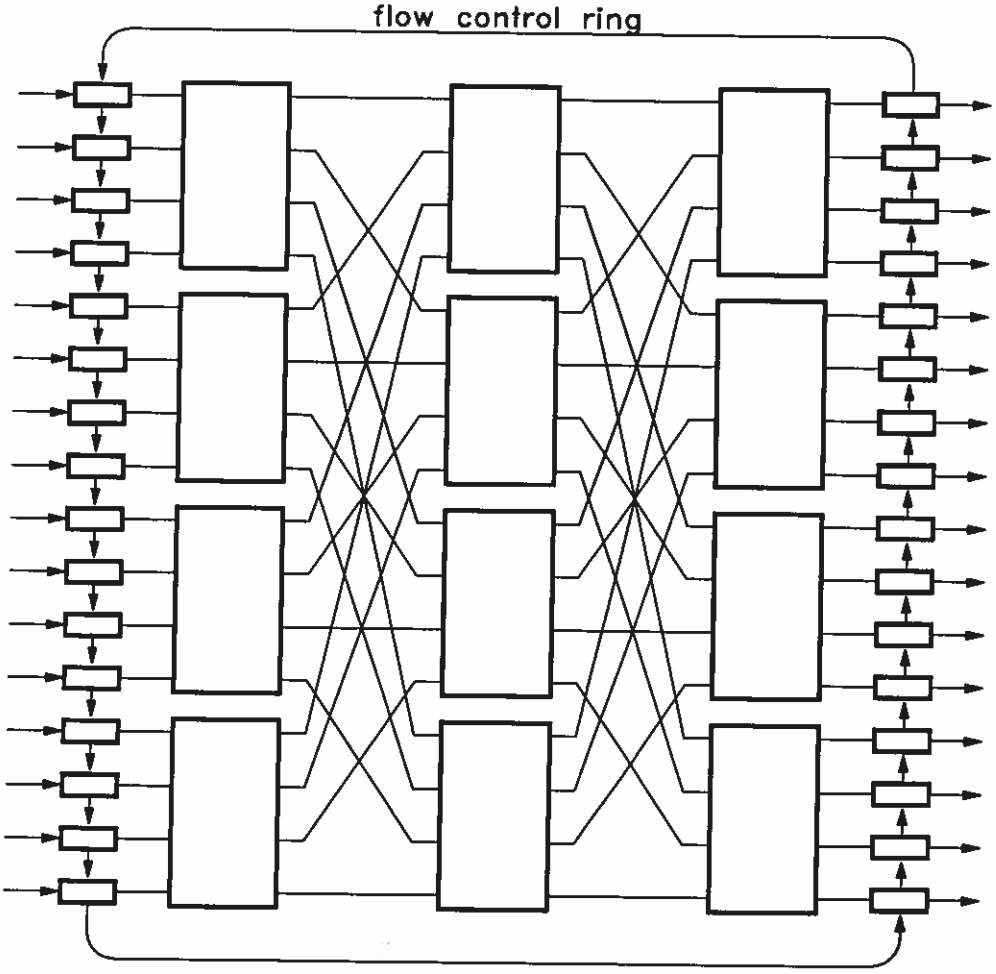

Figure 2.9: Flow Control Across a Switch

each cycle, and it can be made several bits wide if necessary. Switch sizes as large as 256 ports can be easily accommodated if the packet cycle is at least 64 clock ticks long.

We next consider the issue of routing packets in a datagram network. To distribute traffic through the network, we need some method for selecting alternate routes. However, the need for high speed routing translation means that only fairly simple choices can be made on a per packet basis. We present a possible approach that is well-suited to hardware implementation, and which together with the flow control mechanisms described above could be effective. We assume that each switch has a Control Processor (CP) that manages the overall operation of the switch. In particular, the CP maintains a graph model of the entire network including all switches, the capacities of interconnecting trunk groups and possibly

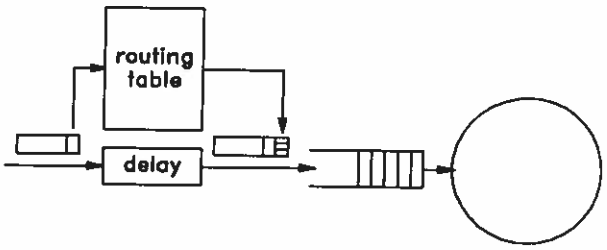

Figure 2.10: Datagram Address Translation

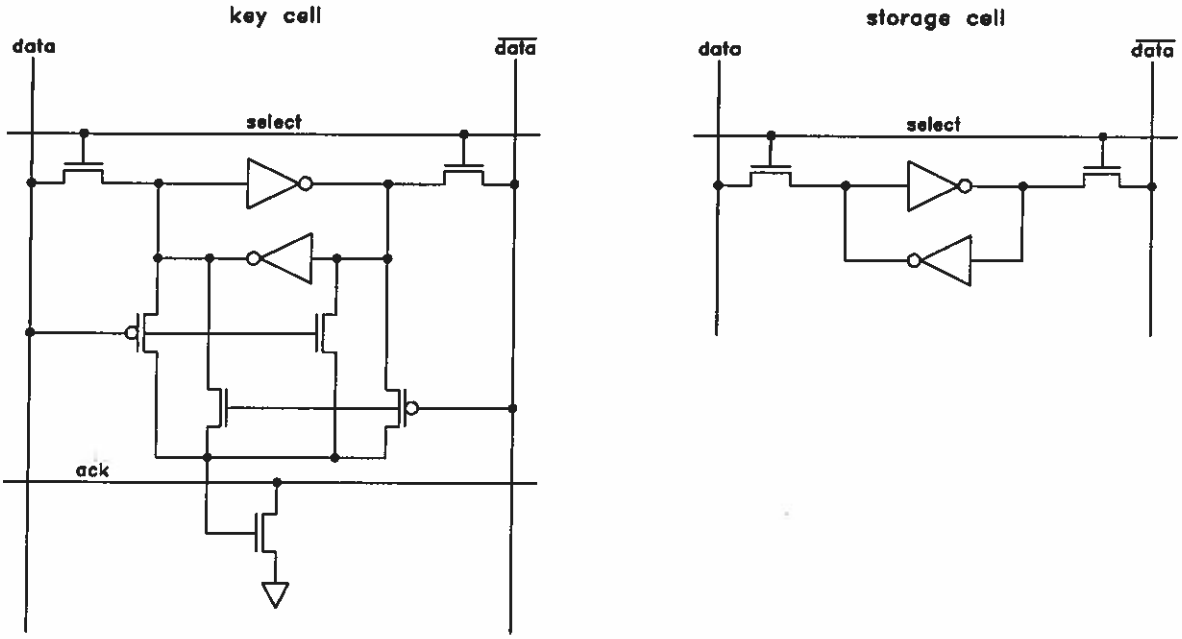

Figure 2.11: Content Addressable Memory Cells

the addresses of all terminals associated with each switch. Using this, the CP can compute for each destination, an ordered list of links that can be used to reach that destination.

Each PP has an address translation table that it can use to translate a network address to a list of links that can be used to reach that address. The content of these tables is determined by the CP. The address translation is done before a packet is placed in an incoming PP's buffer, as shown in Figure 2.10. When the packet reaches the front of the buffer, one of the outgoing links is selected, based on the flow control information communicated through the control ring. That is, if the first outgoing link in a packet's list has not set its flow control bit, the packet is sent to it. If it has set its bit, the second link in the list is tried, and so forth.

There are two obvious approaches to address translation. One is to use a hierarchical address based on physical location that can be translated in several steps. A simple version of this is a two step hierarchy in which the first part of the address identifies the access switch for a particular destination and the second part gives a port number on that switch. In a network with a few thousand switches, the required address translation tables would be small enough to fit readily in a single integrated circuit. Another possibility is the use of a large flat address space. If the address space is very large (say 32 bits), it is impractical to store the translation information for every possible address. We can however reasonably store in each PP the translation information for a few thousand addresses using a content addressable memory (CAM). Figure 2.11 shows a storage cell and a key cell for a CAM. The storage cell is a conventional six transistor memory cell. The key cell contains additional circuitry that allows it to compare the stored bit with the information on the data lines and signal a match using a common precharged acknowledgement line. Notice that the key cell requires 11 transistors meaning that a CAM with 1024 words including a 32 bit key and 32 bits of data would require about 550 thousand transistors, for the main memory array. In one micron CMOS this can be easily fit on a single integrated circuit.

Because a CAM-based address translator cannot store information about all network addresses, we require a mechanism for handling packets for which translation information is not present in the CAM. One approach is to send such packets to the CP. So long as these exceptions occur infrequently, the CP can route them and at the same time update the CAM of the PP where the packet arrived, so that subsequent packets to that destination can be handled by the PP. This of course raises the issue of what happens when a PP's CAM is full. In this case, the CP has to discard one of the entries in the CAM in order to make room for a new one. To determine which entry to discard, the CP needs a way of determining which entries have been used recently. A possible solution to this problem is to supplement each CAM word with a mark bit, which is set by the CAM whenever that word is accessed. The CP can periodically (say every ten seconds) poll the status of all mark bits in the CAM via a control packet and simultaneously clear the mark bits. Using this information it can easily track the entries that have been most recently used.

The use of flow control, large buffers and dynamic packet routing all imply that delays can be large and vary widely when the network is heavily loaded. This in turn suggests that priorities may be required to ensure satisfactory performance for time-critical applications. This is particularly important in the large buffers on the input side of the packet switches, which could have to store as many as several thousand packets (20 Mb buffer with 5000 bit packets). A priority queue controller for such a buffer can be implemented using a circuit similar to the buffer controller described in [99]. We estimate that such a controller would require about 300–500

transistors per packet stored, making a single chip implementation of a controller for a 4096 packet buffer feasible.

3. Nonblocking Multirate Networks

Faculty	Jonathan Turner
Research Associate	Riccardo Melen

In this section we discuss a generalization of the classical theory of nonblocking switching networks to model communication systems designed to carry connections with a multiplicity of data rates. The theory of nonblocking networks was motivated by the problem of designing telephone switching systems capable of connecting any pair of idle terminals, under arbitrary traffic conditions. We extend the classical model to switching systems in which the internal data paths carry multiplexed traffic in which individual user channels consume an arbitrary fraction of the bandwidth, subject only to the constraint that the total traffic not exceed the capacity of the data path. Our model is applicable in particular to packet switched fabrics in which all packets in a given connection follow the same path through the switching system. More details of this work may be found in [60].

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 each input node has one outgoing edge and no incoming edge and each output node has one incoming edge and no outgoing edge. We consider only networks that can be divided into a sequence of *stages*. We say that the 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 the same stage, and no other nodes are in this stage. When we refer to a k stage network, we generally neglect the stages containing the input and output ports. We refer to a network with n input ports and m output ports as an (n, m)-network. We let $X_{n,m}$ denote the network consisting of n input nodes, m output nodes and a single internal node. In this network model, nodes correspond to the hardware devices that perform the actual switching functions and the links to the interconnecting data paths. This differs from the graph model traditionally

used in the theory of switching networks, which can be viewed as a dual to our model.

When describing particular networks we will find it convenient to use a composition operation. We denote the composition of two networks Y_1 and Y_2 by $Y_1 \circ Y_2$. The composition operation yields a new network consisting of one or more copies of Y_1 connected to one or more copies of Y_2, with a link joining each pair of subnetworks. More precisely, if Y_1 has n_1 outputs and Y_2 has n_2 inputs, then $Y_1 \circ Y_2$ is formed by taking n_2 copies of Y_1 numbered from 0 to $n_2 - 1$ followed by n_1 copies of Y_2, numbered from 0 to $n_1 - 1$. Then, for $0 \leq i \leq n_1 - 1$, $0 \leq j \leq n_2 - 1$, we join $Y_1(i)$ to $Y_2(j)$ using a link connecting output port j of $Y_1(i)$ to input port i of $Y_2(j)$. Next, we remove the former input and output nodes that are now internal, identifying the edges incident to them and finally, we renumber the input and output nodes of the network as follows; if u was input port i of $Y_1(j)$, it becomes input $jn_1 + i$ in the new network; similarly if v was output port i of $Y_2(j)$, it becomes output $jn_2 + i$. We also allow composition of more than two networks; the composition $Y_1 \circ Y_2 \circ Y_3$ is obtained by letting $Z_1 = Y_1 \circ Y_2$ and $Z_2 = Y_2 \circ Y_3$, then identifying the copies of Y_2 in Z_1 and Z_2. This requires of course that the number of copies of Y_2 generated by the two initial compositions be the same. Note this is not the same as $(Y_1 \circ Y_2) \circ Y_3$.

A *connection* in a network 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 $V - (I \cup O)$, together with a weight. A route r *realizes* a connection (x, y, ω), if x and y are the input and output nodes joined by r and the weight of r equals ω.

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. A *configuration* for a network G is a set of routes. The weight on an edge in a particular configuration is just the sum of the weights of all routes including that edge. A configuration is compatible if for all edges $(u, v) \in E$, the weight on (u, v) is ≤ 1. A set of connections is said to be *realizable* if there is a compatible configuration that realizes that set of connections. If we are attempting to add a connection (x, y, ω) to an existing configuration, we say that a node u is *accessible* from x if there is path from x to u, all of whose edges have a weight of no more than $1 - \omega$.

A network is said to be *rearrangeably nonblocking* (or simply *rearrangeable*) if for every set C of compatible connections, there exists a compatible configuration that realizes C. A network is *strictly nonblocking* if for every compatible configuration 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 networks, one can choose routes arbitrarily and always be guaranteed that any

new connections can be satisfied without rearrangements. We say that a network is *wide-sense nonblocking* if there exists a routing algorithm, for which the network never blocks; that is, if we use the routing algorithm to select routes for each new connection request, it is always possible to realize a new connection by adding a route to the current configuration.

Sometimes, improved performance can be obtained by placing constraints on the traffic imposed on a network. We will consider two such constraints. First, we restrict the weights of connections to the the interval $[b, B]$. We also limit the sum of the weights of connections involving a node x in $I \cup O$ to β. Note that $0 \leq b \leq B \leq \beta \leq 1$. We say a network is strictly nonblocking for particular values of b, B and β if for all sets of connections for which the connection weights are in $[b, B]$ and the total port weight is β, the network cannot block. The definitions of rearrangeably nonblocking and wide-sense nonblocking networks are extended similarly. The practical effect of a restriction on β is to require that a network's internal data paths operate at a higher speed that the external transmission facilities connecting switching systems, a common technique in the design of high speed systems. The reciprocal of β is commonly referred to as the *speed advantage* for a system.

Two particular choices of parameters are of special interest. We refer to the traffic condition characterized by $B = \beta$, $b = 0$ as unrestricted packet switching (UPS), and the condition $B = b = \beta = 1$ as pure circuit switching (CS). Since the CS case is a special case of the multirate case, we can expect solutions to the general problem to be at least as costly as the CS case and that theorems for the general case should include known results for the CS case.

3.1. Strictly Nonblocking Networks

A three stage Clos [16] network with N input and output ports is denoted by $C_{N,k,m}$, where k and m are parameters, and is defined as: $C_{N,k,m} = X_{k,m} \circ X_{N/k,N/k} \circ X_{m,k}$. A Clos network is depicted in Figure 3.1. The standard reasoning to determine the nonblocking condition (see [16]) can be extended in a straightforward manner, yielding the following theorem.

THEOREM 3.1.1. *The Clos network $C_{N,k,m}$ is strictly nonblocking if*

$$m > 2 \max_{b \leq \omega \leq B} \left\lfloor \frac{\beta k - \omega}{s(\omega)} \right\rfloor$$

where $s(\omega) = \max\{1 - \omega, b\}$.

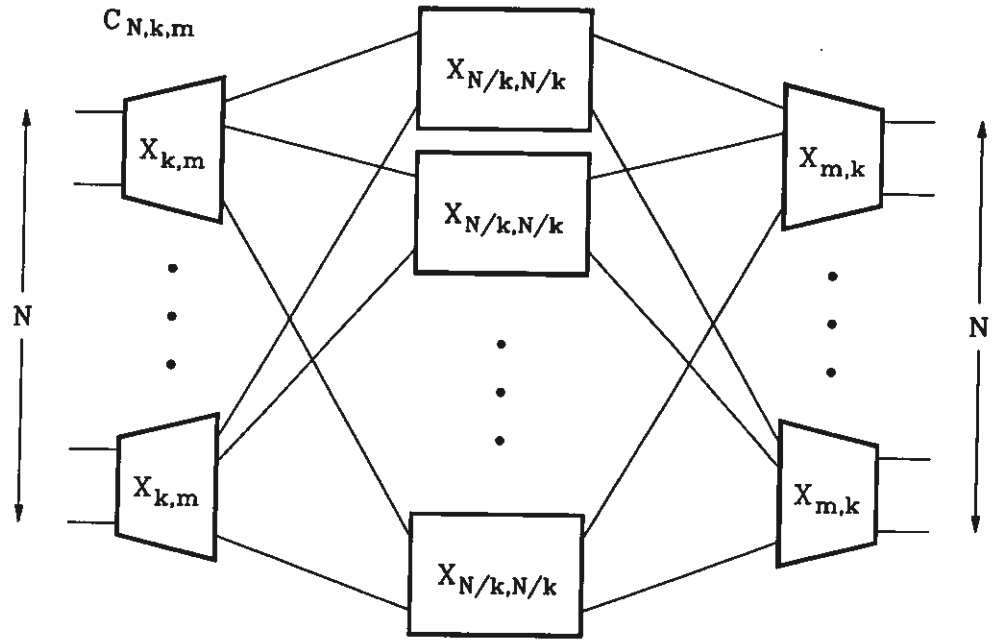

Figure 3.1: Clos Network

Let us examine some special cases of interest. If we let $b = B = \beta = 1$, the effect is to operate the network in CS mode and the theorem states that we get nonblocking operation when $m \geq 2k - 1$ as is well-known. In the UPS case, the condition on m becomes $m > 2(\beta/(1-\beta))(k-1)$. So $m = 2k-1$ is sufficient here also if $\beta = 1/2$.

Using Theorem 3.1.1, we can construct a wide-sense nonblocking network for unrestricted traffic by placing two Clos networks in parallel and segregating connections in the two networks based on weight. In particular if we let $m = 4k-1$, the network $X_{1,2} \circ C_{N,k,m} \circ X_{2,1}$ is wide-sense nonblocking if all connections with weight $\leq 1/2$ are routed through one of the Clos subnetworks and all the connections with weight $> 1/2$ are routed through the other.

A k-ary Beneš network [6], built from $k \times k$ switching elements (where $\log_k N$ is an integer) can be defined recursively as follows: $B_{k,k} = X_{k,k}$ and $B_{N,k} = X_{k,k} \circ B_{N/k,k} \circ X_{k,k}$ (see Figure 3.2). A k-ary Cantor network of multiplicity m is defined as $K_{N,k,m} = X_{1,m} \circ B_{N,k} \circ X_{m,1}$. The next theorem captures the condition on m required to make the Cantor network strictly nonblocking.

THEOREM 3.1.2. *The Cantor network $K_{N,k,m}$ is strictly nonblocking if*

$$m \geq \frac{2\beta}{ks(B)}(1 + (k-1)\log_k(N/k))$$

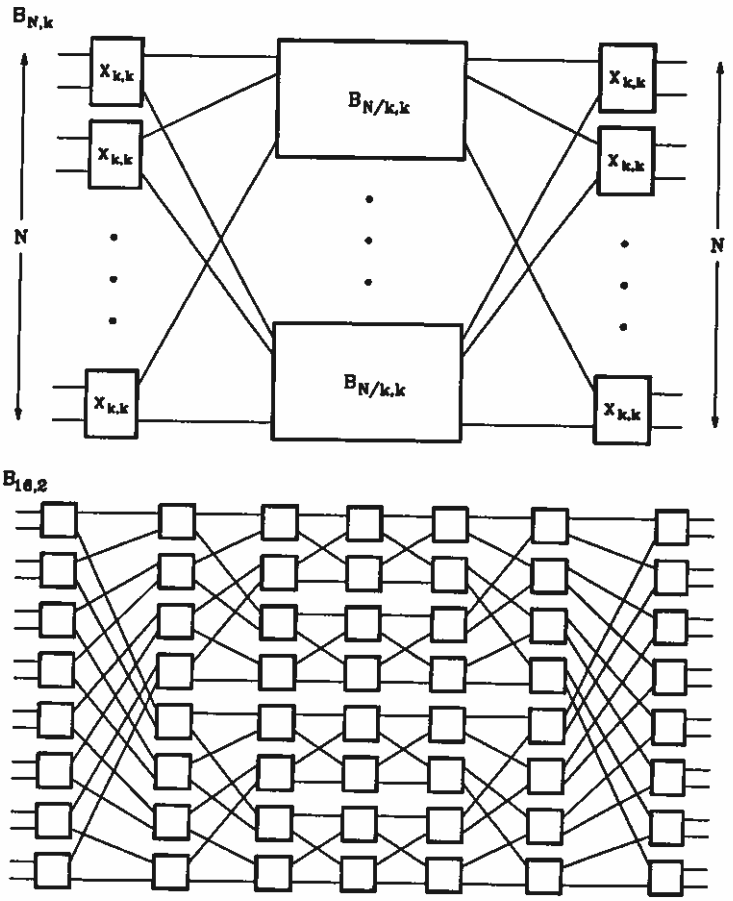

Figure 3.2: Beneš Network ($B_{N,k}$)

When we apply the theorem to the CS case for $k = 2$, we find that the condition on m reduces to $m \geq \log_2 N$ as is well known. For the UPS case with $k = 2$, we have $m \geq (\beta/(1-\beta)) \log_2 N$; that is, we again need a speed advantage of two to match the value of m needed in the CS case. We can construct wide-sense nonblocking networks based on the Cantor network for $\beta = 1$ by increasing m. We divide the connections into two subsets, with all connections of weight $\leq 1/2$

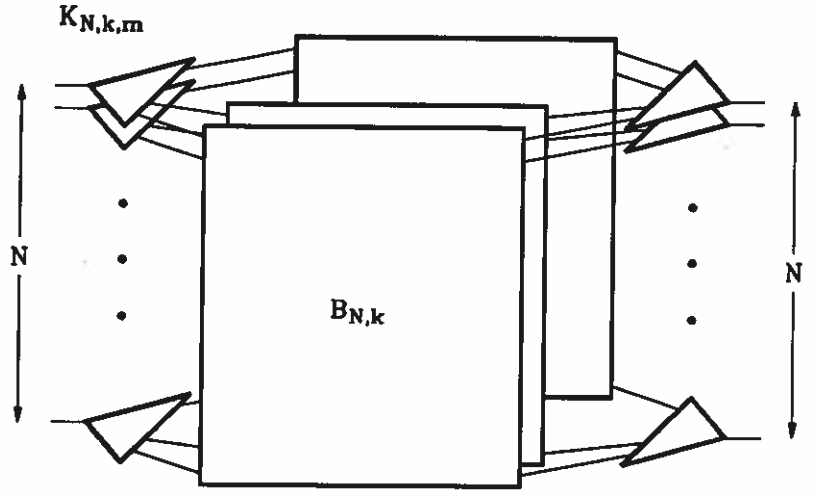

Figure 3.3: Cantor Network

segregated from those with weight $> 1/2$. Applying Theorem 3.1.2 we find that $m \geq 4((k-1)/k)\log_k N$ is sufficient to carry each portion of the traffic, giving a total of $8((k-1)/k)\log_k N$ subnetworks. Using the theorem, we can also obtain the following corollary for the Beneš network.

COROLLARY 3.1.1. *The Beneš network $B_{N,k}$ is strictly nonblocking if*

$$\beta \leq \left\lceil \frac{2}{ks(B)}\left(1 + (k-1)\log_k(N/k)\right)\right\rceil^{-1}$$

So for example, in the UPS case we find that for an r stage Beneš network we require $\beta \leq 1/r$ to get a strictly nonblocking network. For $k = 32$, a three stage Beneš network has 1024 inputs and outputs and requires a speed advantage of three.

3.2. Rearrangeably Nonblocking Networks

As mentioned earlier, a k-ary Beneš network [6], can be defined recursively as follows: $B_{k,k} = X_{k,k}$ and $B_{N,k} = X_{k,k} \circ B_{N/k,k} \circ X_{k,k}$. The Beneš network is rearrangeable in the CS case [6] and efficient algorithms exist to reconfigure it [57, 67]. In this section, we show that under certain conditions, the Beneš network can

be rearrangeable for multirate traffic as well. Our results rely on showing that the graph coloring methods used to route connections in the Beneš network for the circuit switching case can be extended to the multirate case as well.

Our first theorem gives conditions under which this routing is guaranteed not to exceed the capacity of any link in the network.

THEOREM 3.2.1. *$B_{N,k}$ is rearrangeable when*

$$\beta \leq \left[1 + \frac{k-1}{k}(B/\beta)\log_k(N/k)\right]^{-1}$$

As an example, if $N = 1024$, $k = 32$ and $B = \beta$, it suffices to have $\beta \leq 1/2$; for $N = 2^{15}$, it suffices to have $\beta \leq 1/3$. We can improve on this result for binary Beneš networks by modifying the algorithm used to route connections.

THEOREM 3.2.2. *$B_{N,2}$ is rearrangeable when*

$$\beta \leq \left[1 + \frac{1}{4}(B/\beta)\log_2 N\right]^{-1}$$

Theorem 3.2.2 implies for example that if $\beta = B = 0.2$, a binary Beneš network with 2^{16} ports is rearrangeable. Theorem 3.2.1, on the other hand gives rearrangeability in this case only if β is limited to about 0.118. It turns out that we can obtain a still stronger result by a more careful analysis of the original algorithm.

THEOREM 3.2.3. *$B_{N,k}$ is rearrangeable when*

$$\beta \leq [\max\{2, \lambda - \ln\lfloor\beta/B\rfloor\}]^{-1}$$

where $\lambda = 2 + \ln\log_k(N/k)$.

So, for example if $k = 32$, $N = 2^{15}$ and $B = \beta$, we can have $\beta = 0.37$. We now turn our attention to the Cantor network and give conditions for rearrangeability in that case.

THEOREM 3.2.4. *Let $\epsilon > 0$ and $\lfloor\beta/B\rfloor \leq \log_k(N/k)$. $K_{N,k,m}$ is rearrangeable if*

$$m \geq \lceil(1+\epsilon)(\lambda - \ln\lfloor\beta/B\rfloor)\rceil + 2\,(2 + \log_2\lambda + \log_2(B/c))$$

where $\lambda = 2 + \ln\log_k(N/k)$ *and* $c = 1 - \beta\lambda/(1+\epsilon)(\lambda - \ln\lfloor\beta/B\rfloor)$.

So for example, when $k = 32$, $N = 2^{15}$ and $B = \beta = 1/2$, it suffices to have $m = 10$. The graph coloring methods used to route connections for $B_{N,k}$ can also be applied to networks that "expand" at each level of recursion. Let $C^*_{k,k,m} = X_{k,k}$ and for $N = k^i$, $i > 1$, let $C^*_{N,k,m} = X_{k,m} \circ C^*_{N/k,N/k} \circ X_{m,k}$. The following theorem gives conditions under which $C^*_{N,k,m}$ is rearrangeable.

THEOREM 3.2.5. *$C^*_{N,k,m}$ is rearrangeable if*

$$\beta \leq \left[1/\gamma^c + \frac{m-1}{m}\frac{B}{\beta}\frac{1-1/\gamma^c}{1-1/\gamma}\right]^{-1}$$

where $\gamma = m/k$ and $c = \log_k(N/k)$.

So, for example, $C^*_{N,k,2k-1}$ is rearrangeable if $B \leq 1/2$.

3.3. Multipoint Networks

In this section, we describe two multipoint networks and give conditions under which they are rearrangeable or wide-sense nonblocking. A *connection* in a multipoint network is a triple (x, Y, ω) where $x \in I$, $Y \subseteq O$ 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* in a multipoint network is a tree whose root is an input node and whose leaves are output nodes. A route r *realizes* a connection (x, Y, ω), if x and Y are the input and output nodes joined by r and the weight of r equals ω.

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. A *configuration* for a network G is a set of routes. The weight on an edge in a particular configuration is just the sum of the weights of all routes including that edge. A configuration is compatible if for all edges $(u, v) \in E$, the weight on (u, v) is ≤ 1. A set of connections is said to be *realizable* if there is a compatible configuration that realizes that set of connections.

A network is said to be *rearrangeably nonblocking* (or simply *rearrangeable*) if for every set C of compatible connections, there exists a compatible configuration that realizes C. A network is *strictly nonblocking* if for every compatible configuration 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 networks, one can choose routes arbitrarily and always be guaranteed that any new connections can be satisfied without rearrangements. We say that a network is *wide-sense nonblocking* if there exists a routing algorithm, for which the network

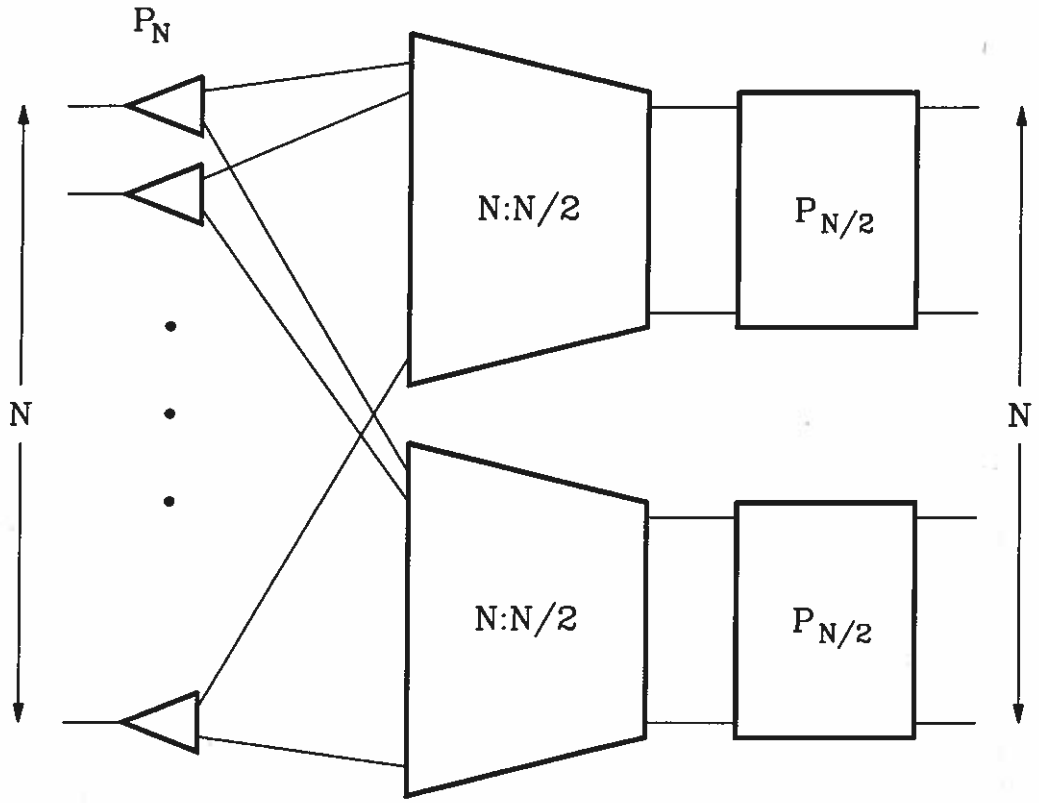

Figure 3.4: Recursive Construction of P_N

never blocks; that is, if we use the routing algorithm to select routes for each new connection request, it is always possible to realize a new connection by adding a route to the current configuration.

The network P_N, shown in Figure 3.4 has a set of N input crossbars, each having one input and two outputs. These feed into a set of *concentrators* with N inputs and $N/2$ outputs. The outputs of the concentrators then feed into recursively constructed networks $P_{N/2}$. The base of the recursive construction is a 2×2 crossbar. We say that a concentrator with n inputs and m outputs ($m \leq n$) is strictly nonblocking for point-to-point connections if given any configuration in which all routes are paths and $< m$ outputs are active, there exists a path connecting any specified idle input to some (unspecified) idle output. We say that such a concentrator is rearrangeable for point-to-point connections if given any set of $\leq m$ inputs, we can find a set of compatible routes connecting the given inputs to some set of m outputs. Pippenger [71] first described this network in 1973 and showed that if the concentrators used are strictly nonblocking for point-to-point connections, then the network as a whole is wide-sense nonblocking for multipoint connections. If Cantor networks are used for concentration, this yields a wide-sense

nonblocking network for multipoint connections with $O(N(\log N)^3)$ complexity. Nassimi and Sahni [64], using the fact that a Banyan network is a rearrangeably nonblocking concentrator for point-to-point connections [87], showed that P_N is rearrangeably nonblocking for multipoint connections when Banyan networks are used for the concentrators. This yields an $O(N(\log N)^2)$ complexity network. We extend these results to the multirate case.

THEOREM 3.3.1. *If the concentrator used to construct P_N is strictly nonblocking for point-to-point traffic, then P_N is wide sense nonblocking for multipoint traffic. If the concentrator used to construct P_N is rearrangeably nonblocking for point-to-point traffic, then P_N is rearrangeably nonblocking for multipoint traffic.*

COROLLARY 3.3.1. *If the concentrator used to construct P_N is the Beneš network $B_{N,k}$ then P_N is wide-sense nonblocking if*

$$\beta \leq \left[\frac{2}{ks(B)}(1+(k-1)\log_k(N/k))\right]^{-1}$$

So for example in the UPS case, if we let $k = 32$, we can construct a wide-sense nonblocking multipoint network if we have $\beta \leq 1/3$.

COROLLARY 3.3.2. *If the concentrator used to construct P_N is a Delta network $D_{N,k}$ then P_N is wide-sense nonblocking if $\beta \leq 1/2$*

The corollary is proved by showing that the Delta network is a rearrangeably nonblocking concentrator for point-to-point multirate connections when $\beta \leq 1/2$.

We can also obtain nonblocking networks by combining two nonblocking point-to-point networks. For example, Thompson [87] has shown that cascading two Banyan networks, yields a rearrangeable network (for the circuit switching case) that can produce any desired number of copies of any specified inputs. If we follow such a network with one that is rearrangeable for point-to-point connections, we get a rearrangeably nonblocking network for multipoint connections. Figure 3.5 illustrates this idea, using a Beneš network to route the copies to their final destinations. The figure also shows how certain pairs of stages can be combined to yield a network with $(4\log_k N) - 3$ stages. We call this network $T_{N,k}$.

We can obtain wide-sense nonblocking networks for multipoint connections in the circuit switching case by cascading two networks that are strictly nonblocking for point-to-point connections.

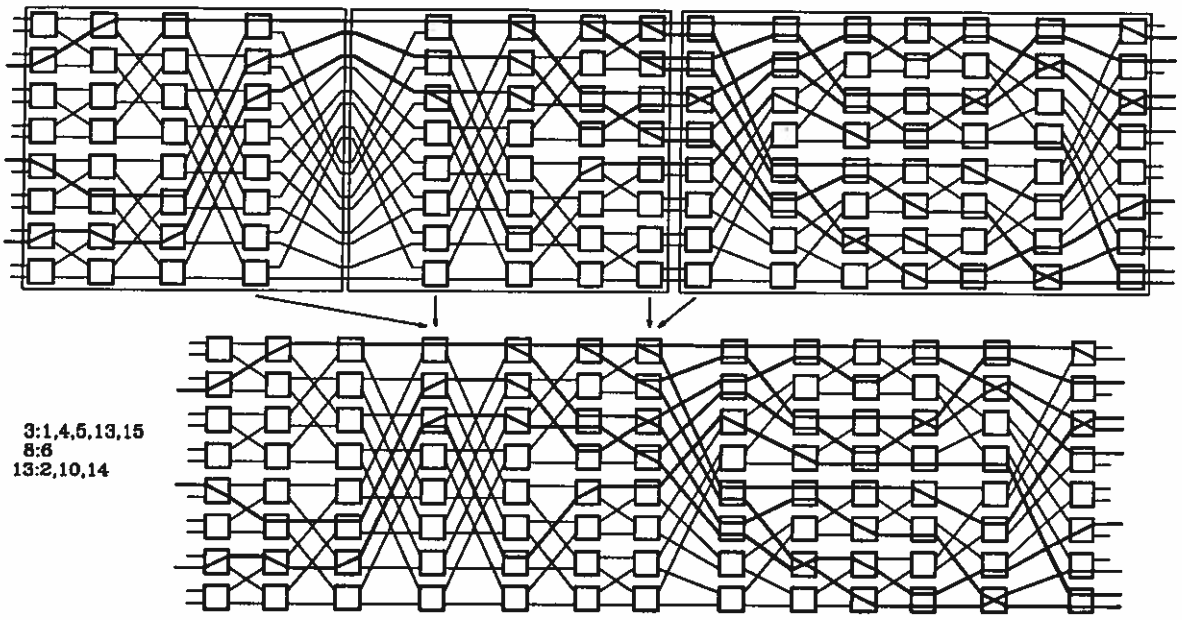

Figure 3.5: Example of Routing in $T_{N,2}$

THEOREM 3.3.2. *A pair of cascaded Clos networks is wide-sense nonblocking for multipoint circuit switched connections. A pair of cascaded Cantor networks is wide-sense nonblocking for multipoint circuit switched connections.*

To obtain nonblocking operation connections must be routed so that branching is performed only in the second network of the pair. So long as this constraint is satisfied, the overall network never blocks. We can extend these results to the multirate case.

THEOREM 3.3.3. *A pair of cascaded Clos networks ($C_{N,k,m}$) is wide-sense nonblocking if*

$$m > 2 \max_{b \le \omega \le B} \left\lfloor \frac{\beta k - \omega}{s(\omega)} \right\rfloor$$

A pair of cascaded Cantor networks ($K_{N,k,m}$) is wide-sense nonblocking if

$$m \ge \frac{2\beta}{ks(B)} (1 + (k-1)\log_k(N/k))$$

$T_{N,k}$ is wide-sense nonblocking if

$$\beta \le \left[\frac{2}{ks(B)} (1 + (k-1)\log_k(N/k)) \right]^{-1}$$

$T_{N,k}$ is rearrangeably nonblocking if

$$\beta \leq [\max\{2, \lambda - \ln\lfloor \beta/B \rfloor\}]^{-1}$$

where $\lambda = 2 + \ln\log_k(N/k)$.

So for example, if we take $k = 32$ and $N = 1024$, $T_{N,k}$ is wide-sense nonblocking for multipoint connections if $\beta \leq 1/3$; for $N = 2^{15}$, we need $\beta \leq 1/5$. For rearrangeable operation we need $\beta \leq 2$ for $N = 1024$ and $\beta \leq .37$ for $N = 2^{15}$.

3.4. Complexity of Multirate Networks

The traditional complexity measure for switching networks is crosspoint count. This measure was motivated by early electromechanical space-division switching systems in which the number of crosspoints in a system was a good indicator of system cost. The technological changes of the last thirty years have led to networks implemented using specialized, high-density integrated circuits, and this traditional complexity measure is no longer closely related to cost. In the current environment, cost is largely determined by the number of integrated circuit components required to implement a system, where the complexity of the components is limited primarily by pin constraints, not by the number of elementary devices within the component.

In the multirate environment, we have found that in order to achieve a certain performance characteristic, it is often necessary to restrict β is some fashion. Practically, this means the data paths within the switch must operate at a higher speed than the external links that the switch interconnects. We can obtain such higher speeds either by increasing the internal clock frequency or by increasing the parallelism in the data paths. If one is to compare the complexity of different networks in order to assess design alternatives, one must assume that parameters such as clock frequency are the same. Consequently, if one network requires higher speed data paths than another, that higher speed must be obtained through parallelism. Since the cost of the system scales fairly directly with the amount of parallelism, we can account for differences in β by making our complexity measure inversely proportional to β.

Based on these considerations, we define the *complexity* C of a network with N input and output nodes to be the number of components required to construct the network divided by βN, where the number of signal pins in each component is limited to some constant κ. For current technology, realistic values for κ are in the range of 50–100.

For example, the complexity of $B_{N,k}$ is given by

$$\mathcal{C}(B_{N,k}) = (1/k\beta)\log_k(N/k)$$

If we take

$$\beta \leq \left[\frac{2}{ks(B)}(1+(k-1)\log_k(N/k))\right]^{-1}$$

which gives strictly nonblocking operation and let $b = 0$, $B = \beta \leq 1/2$, we obtain a complexity of at most

$$\frac{4}{k^2}(1+(k-1)\log_k(N/k))\log_k(N/k)$$

To account for the pin constraint we take $k = \kappa/2$. So for example, the complexity of a strictly nonblocking Beneš network with 1024 ports and $\kappa = 64$ is .125 components per port; that is, we can construct such a network with 128 components. If we take $N = 2^{15}$ we obtain 0.5 components per port. If we are interested only in rearrangeably nonblocking operation we can take

$$\beta \leq [\max\{2, 2+\ln\log_k(N/k) - \ln\lfloor\beta/B\rfloor\}]^{-1}$$

If we take $b = 0$, $B = \beta$, the complexity is

$$(1/k)(2+\ln\log_k(N/k))\log_k(N/k)$$

If we take $k = \kappa/2 = 32$ and $N = 1024$ this is .0625 components per port. If we take $N = 2^{15}$ we have about .17 components per port.

For networks of restricted depth such as the three stage Clos network, the situation is somewhat more complicated, since in this case we may have to construct the basic switching elements from which such networks are constructed from many smaller components satisfying the pin constraint κ. We take the complexity of a switching element with k_1 inputs and k_2 outputs where $k_1 + k_2 > \kappa$ to be $(4k_1k_2/\kappa^2)$ since we can construct such an element from this many components. The complexity of the Clos network is then

$$\mathcal{C}(C_{N,k,m}) = \frac{4m}{\beta\kappa^2}\left[2+N/k^2\right]$$

If we take let $b = 0$, $B = \beta = 1/2$, let $m = 2k$ which gives strictly nonblocking operation and $k = \sqrt{N/2}$, we obtain a complexity of $\frac{64}{\kappa^2}\sqrt{N/2}$ If we take $\kappa = 64$ and $N = 1024$, we obtain .35 components per port. If we take $N = 2^{15}$ we obtain 2 components per port.

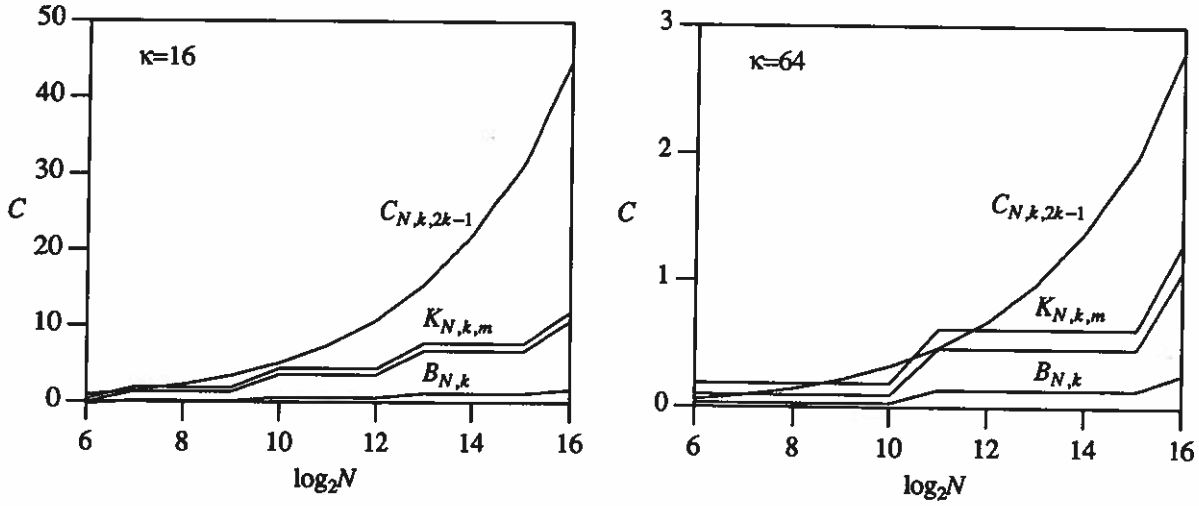

Figure 3.6: Complexity of Multirate Networks: Beneš is shown for rearrangeable and strictly nonblocking case. $\beta = 1/2$ for Clos and Kantor.

For the Cantor network,

$$\mathcal{C}(K_{N,k,m}) = (2(m+1)/\kappa) + (m/k\beta)\log_k(N/k)$$

If we let $b = 0$, $B = \beta = 1/2$, let m be just large enough to give strictly nonblocking operation, and $k = \kappa/2$ this becomes

$$(1/k) + (4/k^2)\,(1 + (k-1)\log_k(N/k))\,((1/2) + \log_k(N/k))$$

Taking $\kappa = 64$ and $N = 1024$ yields about .22 components per port. Taking $N = 2^{15}$ yields about .65 components per port.

The plots in Figure 3.6 compare the complexity of several point-to-point networks across a range of sizes and for two different pin constraints.

4. Prototype Hardware Design

Faculty	Jonathan Turner
Research Associate	Pierre Costa
Graduate Students	Neil Barrett
	Gaurav Garg
	Tony Mazraani
	George Robbert
	Einir Valdimarsson

Considerable progress has been made on the Broadcast Packet Switch prototype during the past year. 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). The CN and RN are composed of binary *Packet Switch Elements* (PSE) that perform routing, traffic distribution and packet replication. A set of *Packet Processors* (PP) provides 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.

Custom integrated circuits are being designed for the PSEs, BTCs and PPs. The BTC and PSE designs require one chip apiece, the PP requires four chips, meaning that a total of 60 custom chips are needed to implement the prototype switch module.

The first chips we designed, implemented the PSE and BTC using four bit wide data paths. These chips have been fabricated and tested. Based on the results of these initial designs we have made revisions that we expect will substantially improve the performance, allowing us to meet our objective of supporting 150 Mb/s transmission links. The design of the eight bit PSE chip has been completed and

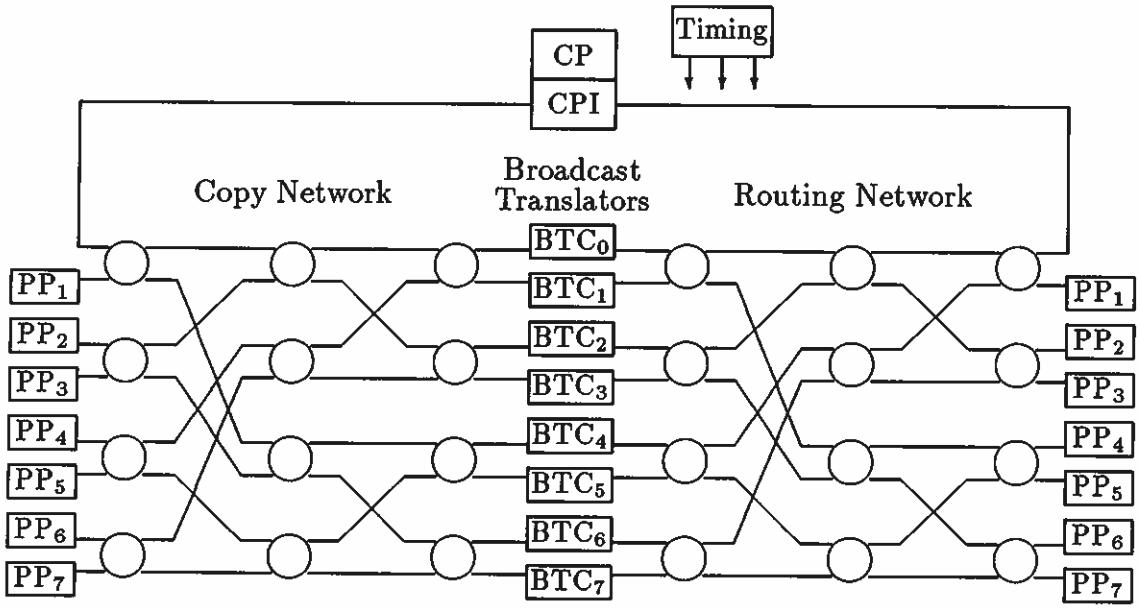

Figure 4.1: Prototype Switch Module

that chip is now being fabricated. The design of the packet buffers to be used in the PP was also completed and these chips will be submitted for fabrication in the near future. The design of a lookup table that will be used for logical channel translation and broadcast channel translation has been completed. The circuit generator for synchronous streams processors is being used to generate major portions of the PP and BTC circuits and has allowed these designs to be generated in a fraction of the time that would have been required using manual methods. The design of the chips implementing these components should be complete before the end of this year.

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. Figure 4.2 depicts the packet formats. A brief description is given below. Another representation of the packet formats is shown in Figure 4.3. This shows the type definitions used by our automatic circuit generator. This will be discussed in more detail below.

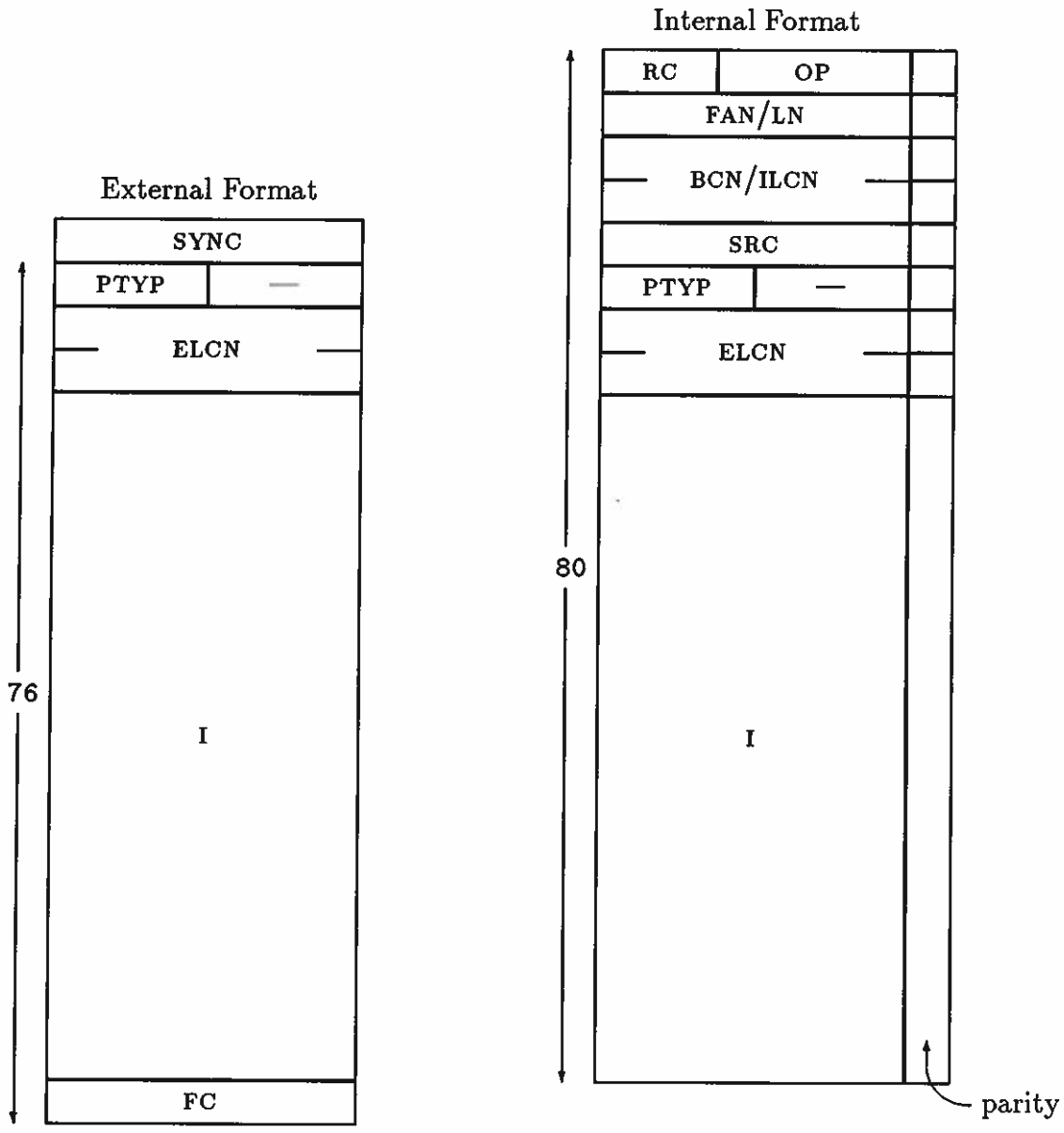

Figure 4.2: Packet Formats

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 (1), test packets (2) and control packets (4).
- *External Logical Channel Number* (ELCN). Logical channel numbers are used to identify which connection a packet belongs to. For the prototype, 64 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], ... 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 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.

```
                        /* external packet types */
#define DATA 1          /* ordinary data packet */
#define LTEST 2         /* link test packet */
#define CTL 4           /* control packet */

                        /* routing control types */
#define IDLE 0          /* unused packet slot */
#define PPNT 1          /* point to point packet */
#define MPNT 2          /* multipoint */
#define SPATH 4         /* specific path */
#define PENDING 7       /* waiting for logical channel translation */

                        /* internal packet op codes */
#define VANILLA 0       /* no control functions */
#define RLCXT 1         /* read LCXT entry */
#define WLCXT 2         /* write LCXT entry */
#define RBTT 3          /* read broadcast translation table entry */
#define WBTT 4          /* write broadcast translation table entry */
#define STEST1 5        /* switch test, first leg */
#define STEST2 6        /* switch test, second leg */
#define STEST3 7        /* switch test, third leg */

typedef struct {                    /* external packet */
        bit         fill[5];        /* unused */
        bit         ptyp[3];        /* packet type */
        bit         elcn[16];       /* external logical channel number */
        bit         info[72][8];    /* information field */
} ext_pkt;

typedef struct {                    /* internal packet */
        bit         op[5];          /* op field */
        bit         rc[3];          /* routing control field */
        bit         fan_ln[8];      /* fanout/ln */
        bit         bcn_ilcn[16];   /* bcn/lcn */
        bit         src[8];         /* source of packet */
        ext_pkt         extp;
} int_pkt;
```

Figure 4.3: Packet Format Definitions

0 *Vanilla Packet.* No control functions.

1 *Read* LCXT *Entry.* Directs PP to read a single entry block from the Logical Channel Translation Table. I[0] specifies which block to read. The data is copied into I[1]–I[4] and the packet is returned to the CP.

2 *Write* LCXT *Entry.* Directs PP to write a single entry to the Logical Channel Translation Table. I[0] specifies the block to write. The data to be written is in I[1]–I[4].

6 *Read* BTT *Entry.* Directs BTC to read and return a single entry from the Broadcast Translation Table (BTT). I[0] field specifies which block to read. The data is copied into I[1]–I[4] and the packet returned to the CP.

7 *Write* BTT *Entry.* Directs BTC to write information into a single entry of the BTT. I[0] field specifies which block to write. The data to be written is in I[1]–I[4].

5 *Switch Test Packet, First Leg.* When received by a PP is returned to the SF with the RC field changed to 4 (specific path), the OP field changed to 6 (for second leg) and a new routing field. The new routing field is obtained by rotating bytes 1–4, by one byte position; that is byte 1 becomes byte 4, bytes 2-4 become bytes 1–3.

6 *Switch Test Packet, Second Leg.* When received by a PP is returned to the SF with the RC field changed to 0 (vanilla), the OP field changed to 7 (for third leg) and the LN set to 0 (for the CP).

7 *Switch Test Packet, Third Leg.* A switch test packet being returned to CP.

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. 64 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 86 clock times. This allows time for one packet to be processed and leaves a guard time of six clock periods between packets.

All the custom integrated circuits designed for the prototype use a two phase non-overlapping clock. The two phases are called `phi1` (ϕ_1) and `phi2` (ϕ_2) and are shown in Figure 4.4. For this prototype, the target clock period is 24 ns. The two clock phases are asymmetric, as the various circuits are designed to gate signals only on `phi1`. As shown in the figure, there is a second set of signals `phi1*` and `phi2*` that have the same pulse widths as the main clocks but skip every other cycle. These clocks are used in the transmit portion of the packet processors. The global timing generator provides the global clock signals that drive the system plus a set of signals that define various instants within the global time reference. The notation `gt`X is used to denote clock cycle X in the global time reference. Figure 4.4 shows the waveform for a typical signal `gt`X. Note that the signal level changes when `phi2` is high and is stable when `phi1` is high. By definition, `gt0` is the time at which packets leave the packet processor on their way to the first stage of the copy network. The nodes of the switch fabric delay packets passing through them for exactly 24 clock ticks and the BTC delays packets for exactly 32 clock ticks. In addition, there are two clock ticks of delay whenever signals cross between circuit boards. Figure 4.5 shows the times at which packets pass between

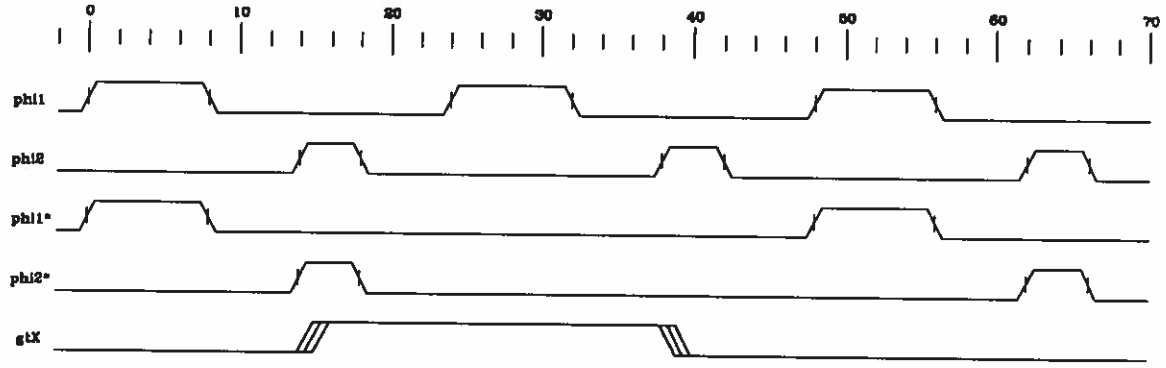

Figure 4.4: Global Clock Signals

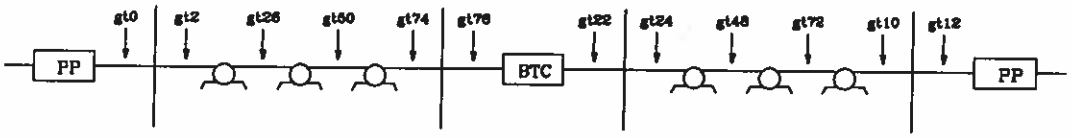

Figure 4.5: Global Timing Relationships

the various components of the system. In this diagram, the vertical lines indicate circuit board boundaries.

The transmit portion of the outgoing packet processor requires one additional signal, called **even**. This signal is asserted throughout every second epoch. That is, it goes high just before **gt0**, stays high for 86 clock ticks and is then low for the next 86 clock ticks. We extend our notation for global time instants as follows. We define **gt0*** to be the clock ticks when **gt0** and **even** are both high and we define **gt**X* to be clock tick X of the slow clock **phi***. So for example **gt3*** occurs whenever **gt6** and **even** are both high. Similarly, **gt80*** occurs when **gt74** is high and **even** is low.

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 **t0** 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 in Figure 4.5. Because the timing circuits used to create the local timing signals have an internal delay of two clock ticks, the synchronization signals that start their local timing cycle must arrive two clock ticks before the logical **t0** time. These signals are usually called **tm2**, which stands for "time minus 2."

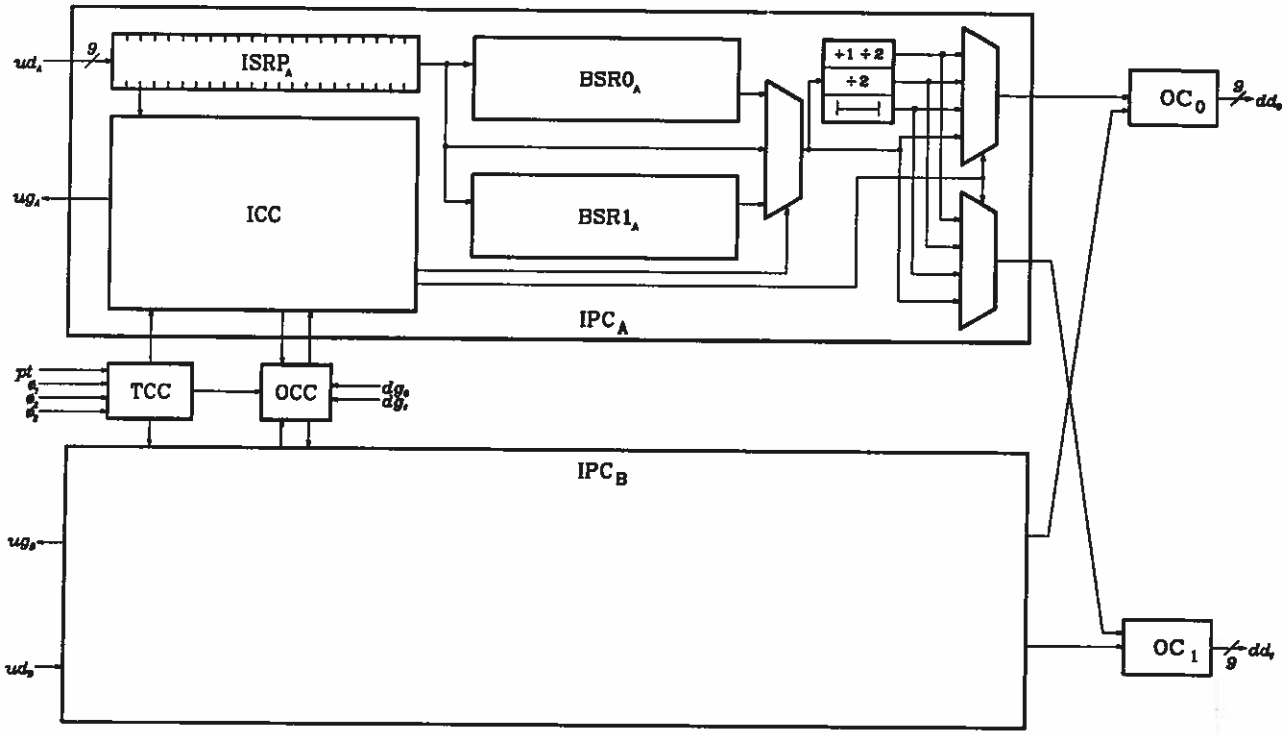

Figure 4.6: Block Diagram of Packet Switch Element Chip

4.3. Packet Switch Element

The Packet Switch Element chip (PSE) is the 2×2 VLSI switch element used in the 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.

An eight bit version of the PSE has been designed during the past year. This version eliminates constraints on the speed of operation of the PSE that were present in the earlier four bit design. This improvement is due largely to changes in some basic design decisions. The most important change was a modification of the way grant propagation was handled. In the system as described in [93], 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. The new design was completed in this summer and simulated at clock speeds of 40 Mb/s. Given the eight bit data path widths, this is fast enough to support data rates of 320 Mb/s, which is slightly faster than needed for the prototype.

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 =route; use bit sn of the LN field to select an output port, where sn is the stage number.
- For om =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 =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 t0 and in general, the clock period during which word i appears is called ti. The delay through a node is 24 clock ticks. So, if an incoming packet can be switched through a node without buffering, the first byte will appear on the output at t24. Each node makes its upstream grant signals available at t18 in the node's frame of reference and holds the grant leads in that state until t18 of the subsequent cycle. Consequently, the grant signal is available to the upstream neighbor any time after t42 in the neighbor's frame of reference.

A block diagram of the PSE appears in Figure 4.6. 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 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 rn, r1, and r0 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 Port Circuits* (IPCA,IPCB). There is one input circuit for each input port. Each IPC 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 and Control Circuit* (TCC). This circuit generates signals of the form ti and ti:j, for various values of i, j. Signal ti is high during clock period ti of the epoch; in particular it goes high during phi2 of the preceding clock cycle and goes low before phi2 goes high again. Signal ti:j is similar; it is high during ti and stays high through tj.

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

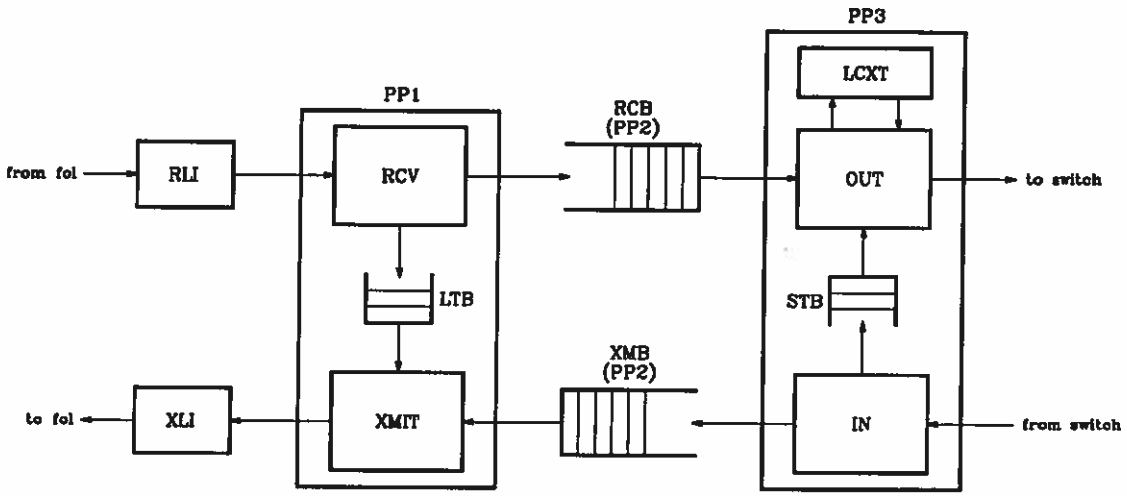

Figure 4.7: Packet Processor Block Diagram

level protocol functions, including the determination of how packets are routed. A block diagram of the PP appears in Figure 4.7. The major 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 and XMB each have a capacity of 32 packets. The LTB and STB can each hold two packets. Together, the four buffers require a total of about 46 Kbits of memory.

- *Receive Link Interface* (RLI). Converts the incoming optical signal to an eight bit electrical format, and provides a clock recovered from the incoming data stream.

- The *Receive Circuit* (RCV). Checks incoming packets for errors, adds parity, strips off FC, routes test packets to the LTB and other packets to the RCB.

- *Output Circuit* (OUT). Adds five bytes of header information to the front of each packet received from the RCB. Performs logical channel translation and sends packets to the SF. Also reads switch test packets and LCXT read/write packets from the STB and processes them appropriately.

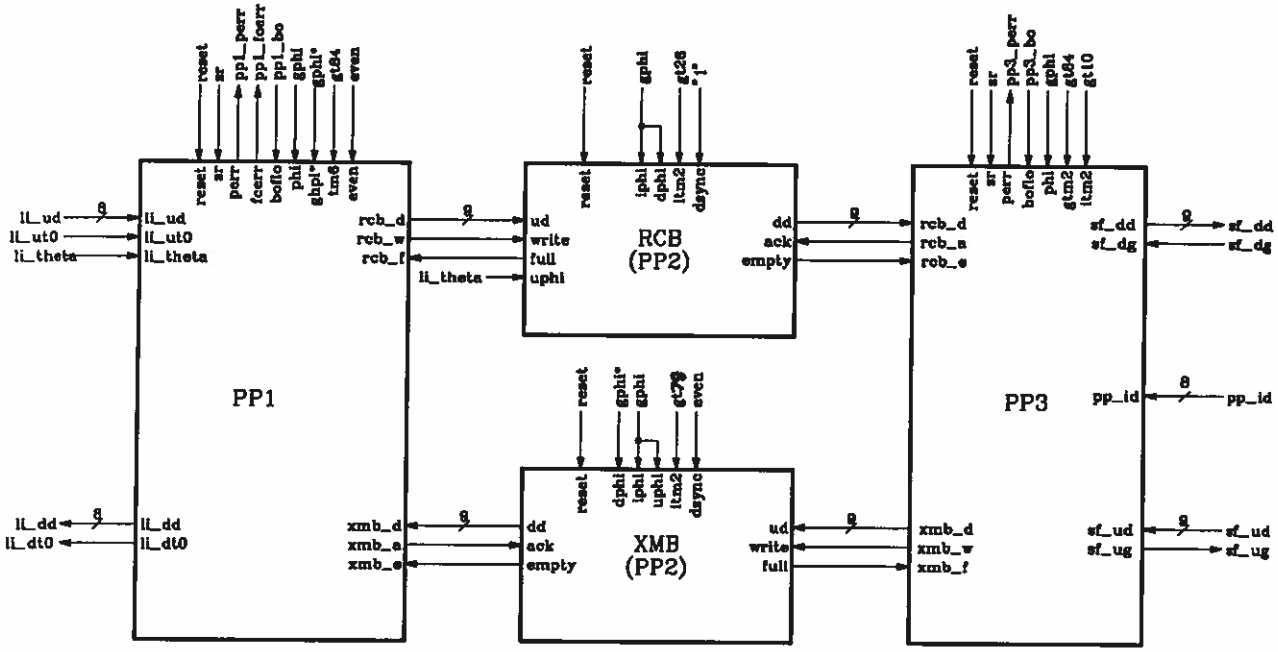

Figure 4.8: Packet Processor Signals

- *Logical Channel Translation Table* (LCXT). Lookup table used to translate an incoming logical channel number to the routing information needed by the switch fabric.
- *Input Circuit* (IN). Routes internal data packets to the XMB, removing the first five bytes of header information and routes all other packets to the LTB.
- *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.

Note that the figure also shows how the components are divided among the different integrated circuits. The RCV and XMIT circuits, together with the LTB will be placed on the PP1 chip, the RCB and XMB each consist of a PP2 chip and the IN and OUT circuits together with the STB and LCXT will be placed on the PP3 chip. The RLI and XLI are being implemented using standard components. The buffers are described in some detail in the next section. Details of the PP1 and PP3 chips appear below.

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 control 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 processing is described above under packet formats.

Figure 4.8 shows the signals connecting to the PP and between its different parts. The major external signals are summarized briefly below.

- *Upstream data from SF* (`sf_ud`). Data from switch fabric. Nine bits wide including parity.
- *Downstream data to SF* (`sf_dd`). Data to switch fabric. Nine bits wide including parity.
- *Downstream grant from SF* (`sf_dg`). When asserted, allows PP to transmit packet in subsequent epoch.
- *Upstream grant to SF* (`sf_ug`). Asserted by PP to allow SF to transmit packet. In prototype, always asserted.
- *Data from link* (`li_ud`). Data stream from FOL. Eight bits wide.
- *Incoming Link framing* (`li_ut0`). Link framing signal. High when first byte of packet is sent.
- *Link clock* (`li_theta`). Clock signal recovered from link.
- *Data to link* (`li_dd`) Data stream to FOL. Eight bits wide.
- *Outgoing Link framing* (`li_ut0`). Outgoing link framing signal. Held high during transmission of a packet.

- PP *identifier* (pp_id). 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.
- *PP1 buffer overflow* (pp1_bo). Asserted whenever a packet is discarded by PP1 due to buffer overflow.
- *PP3 buffer overflow* (pp3_bo). Asserted whenever a packet is discarded by PP3 due to buffer overflow.
- *FC error* (pp1_fcerr). Asserted when the PP receives a packet containing a bad frame check field.
- *Parity error* (pp1_perr, pp3_perr). This signal is asserted whenever the PP detects a parity error.
- *Even epoch* (even). Asserted during every other global epoch.

A block diagram of the PP1 chip appears in Figure 4.9. Packets from the incoming link are checked for framing errors and steered by the RCV circuit to either the LTB or RCB. The XMIT circuit takes packets from either the XMB or LTB and forwards them to the outgoing link. The *control and timing circuit* (CTL/TIM) synchronizes the various components. The RCV circuit and the input half of the LTB operate using the clock recovered from the incoming fiber optic link. The internal portion of the LTB operates using the main system clock (gphi) and the output portion of the LTB along with the XMIT circuit use a half speed version of the main system clock (gphi*).

Figure 4.10 shows a program that specifies the functions of the RCV circuit. A similar specification has been written for the XMIT circuit. The *synchronous streams processor compiler* (*sspc*) produces a circuit from such a specification. The area of the circuits generated for the RCV and XMITcircuits is 3 mm^2 and 2 mm^2 respectively (in 2 micron CMOS). The control and timing circuit will also be generated using a circuit generator that was developed for this purpose, as will the LTB. Because the major components of the PP1 can all be generated automatically, we anticipate the layout of this chip to be completed by the end of October and that the chip will be ready for fabrication before the end of the year.

Figure 4.11 is a block diagram of the PP3 chip. Packets entering the chip from the RCB, pass through the OUT1 circuit, a delay line and the OUT2 circuit before passing on to the switch fabric. When such a packet passes through the

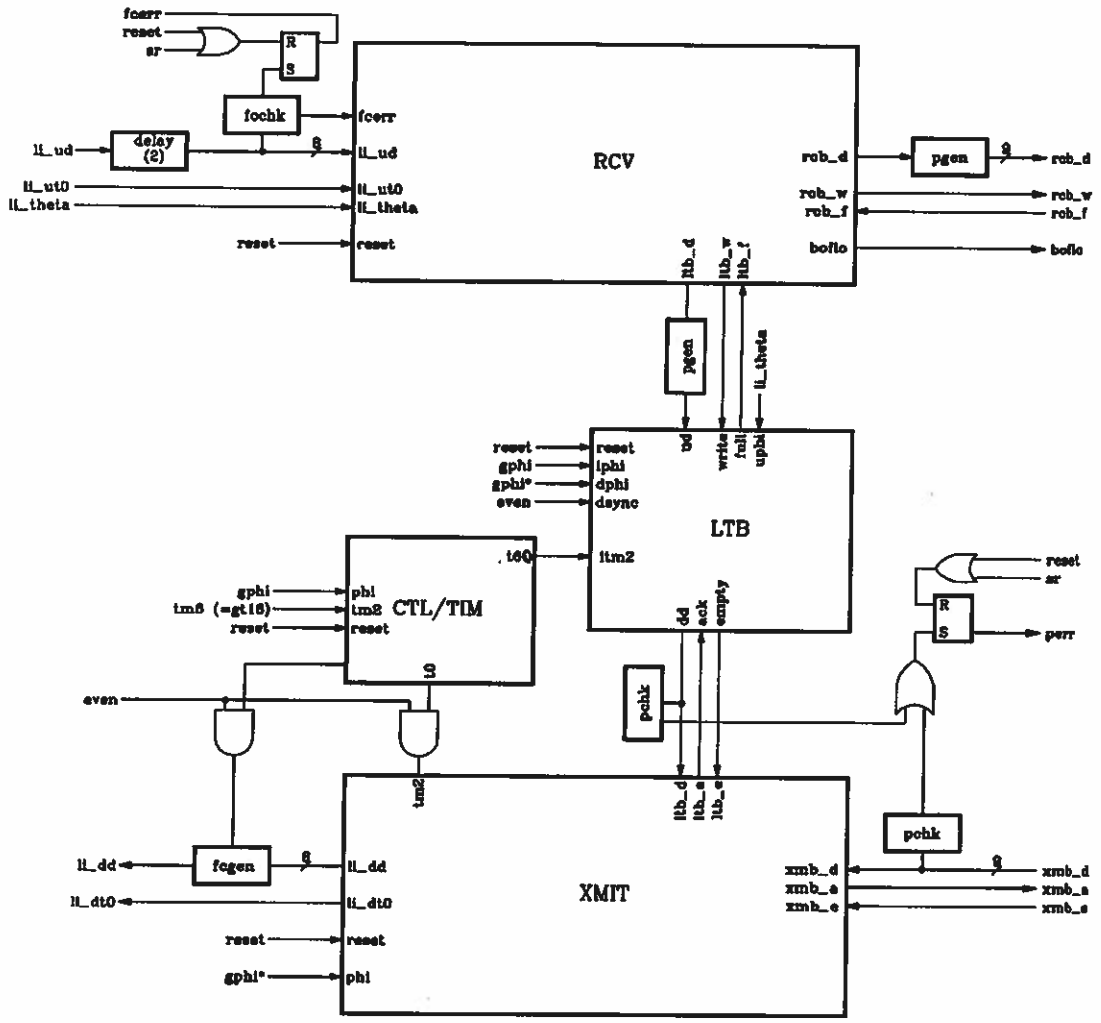

Figure 4.9: PP1 Block Diagram

OUT1 circuit, its logical channel number is passed to the LCXT, which extracts the selected entry and delivers it to the OUT2 circuit in time to be inserted into the packet. This division of the ouput processing functions into two parts was necessitated by limitations in the current implementation of *sspc*. The OUT1 and OUT2 circuits also cooperatively process switch test packets and LCXT read and write packets. Packets coming from the switch fabric pass through the INC circuit and are steered to either the STB or XMB. Packets going to the XMB are stripped of the additional header information added when the packet was first received.

```
rcv086( port[8] ext_pkt <li_ud00;        /* data from link interface */

        port[8] ext_pkt >rcb_d04;        /* data to rcb              */
        port[1] bit     >rcb_w080;       /* rcb write signal         */
        port[1] bit     <rcb_f00;        /* rcb full signal          */

        port[8] ext_pkt >ltb_d04;        /* data to ltb              */
        port[1] bit     >ltb_w080;       /* ltb write signal         */
        port[1] bit     <ltb_f00;        /* ltb full signal          */

        port[1] bit     <fcerr075;       /* frame check error        */
        port[1] bit     >boflo080)       /* buffer overflow          */
{
        boflo = 0;
        rcb_w = 0; rcb_d = li_ud;
        ltb_w = 0; ltb_d = li_ud;
        if li_ud.ptyp == LTEST ->
                ltb_d.ptyp = CTL;
                if ltb_f == 1 & fcerr == 0 ->
                          boflo = 1;
                 | ltb_f == 0 & fcerr == 0 ->
                          ltb_w = 1;
                fi;
        |  li_ud.ptyp == DATA | li_ud.ptyp == CTL ->
                if rcb_f == 1 & fcerr == 0->
                        boflo = 1;
                 | rcb_f == 0 & fcerr == 0 ->
                        rcb_w = 1;
                fi;
        fi;
}
```

Figure 4.10: SSP Program for Receive Circuit

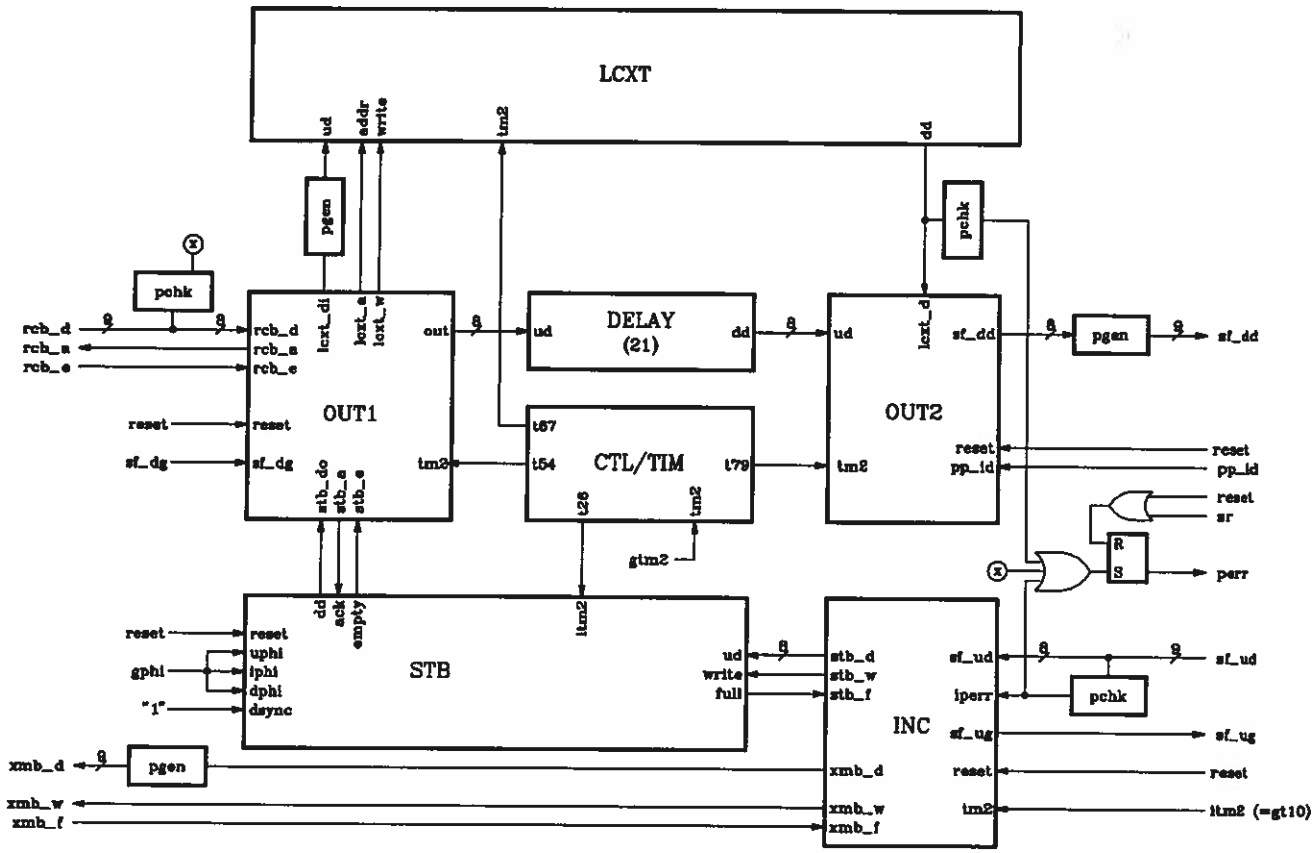

Figure 4.11: PP3 Block Diagram

```
out1@86(port[8] ext_pkt     <rcb_d@0;      /* data from rcb            */
        port[1] bit         <rcb_e@0;      /* rcb empty signal         */
        port[1] bit[16]     >rcb_a@8;      /* rcb ack signal           */

        port[8] int_pkt     <stb_d@0;      /* data from stb            */
        port[1] bit         <stb_e@0;      /* stb empty signal         */
        port[1] bit[16]     >stb_a@8;      /* stb ack signal           */

        port[8] int_pkt     >dd@4;         /* downstream data          */

        port[8] bit[8]      >lcxt_a@12;    /* lcxt address lines       */
        port[8] bit[4][8]   >lcxt_di@13;   /* lcxt data lines          */
        port[1] bit[16]     >lcxt_w@6;     /* lcxt write signal        */

        port[1] bit         <sf_dg@0)      /* grant from switch fabric */
{
        dd.rc = 0;
        rcb_a = 0;
        stb_a = 0;

        lcxt_w = 0;
        lcxt_a = stb_d.extp.info[0];
        lcxt_di:(0..31) = stb_d.extp.info:(8..39);

        if sf_dg == 1 & rcb_e == 0 & stb_e == 1 ->
                dd.extp = rcb_d;
                rcb_a = 0xffff;
                if rcb_d.ptyp == DATA ->          /* start logical channel */
                        dd.rc = PENDING;          /* translation           */
                        lcxt_a = rcb_d.elcn:(0..7);
                 | rcb_d.ptyp != DATA ->          /* route non-data packets */
                        dd.rc = PPNT;             /* to CP.                */
                        dd.fan_ln = 0;
                fi;
        | sf_dg == 1 & stb_e == 0 ->
                stb_a = 0xffff;
                if stb_d.op == RLCXT ->           /* start lcxt read */
                        dd = stb_d;
                 | stb_d.op == WLCXT ->           /* write lcxt */
                        lcxt_w = 0xffff;
                 | stb_d.op == STEST1 | stb_d.op == STEST2 ->
                        dd = stb_d;               /* out2 does rest */
                fi;
        fi;
}
```

Figure 4.12: SSP Program for Out1 Circuit

Figure 4.12 gives the specification defining the functions of the OUT1 circuit. Similar specifications have been written for the OUT2 and INC circuits. The area of the generated circuits is 2.4 mm^2, for the INC circuit, 6.6 mm^2 for the OUT1 circuit and 4.4 mm^2 for the OUT2 circuit. The STB LCXT DELAY, and CTL/TIM circuits are all being created using circuit generators that have been developed for that purpose. We anticipate that layout of this chip will be completed by the end of October and that it will be ready for fabrication by the end of the year.

4.5. Broadcast Translation Circuit

The Broadcast Translation Circuit (BTC) provides unique addresses for each of the copies of a broadcast packet replicated by the copy network. A block diagram of the BTC is shown in Figure 4.13. The major signals interfacing to the BTC are 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 BTC error flag.
- *Parity Error* (**perr**). Asserted when the BTC detects a parity error.
- *Start of epoch* (**gtm4**). Goes high four ticks before the first byte of packet is present on udleads.

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 *Entry.* This two packet types are used for updating the BTT and for reading it for auditing and testing purposes.

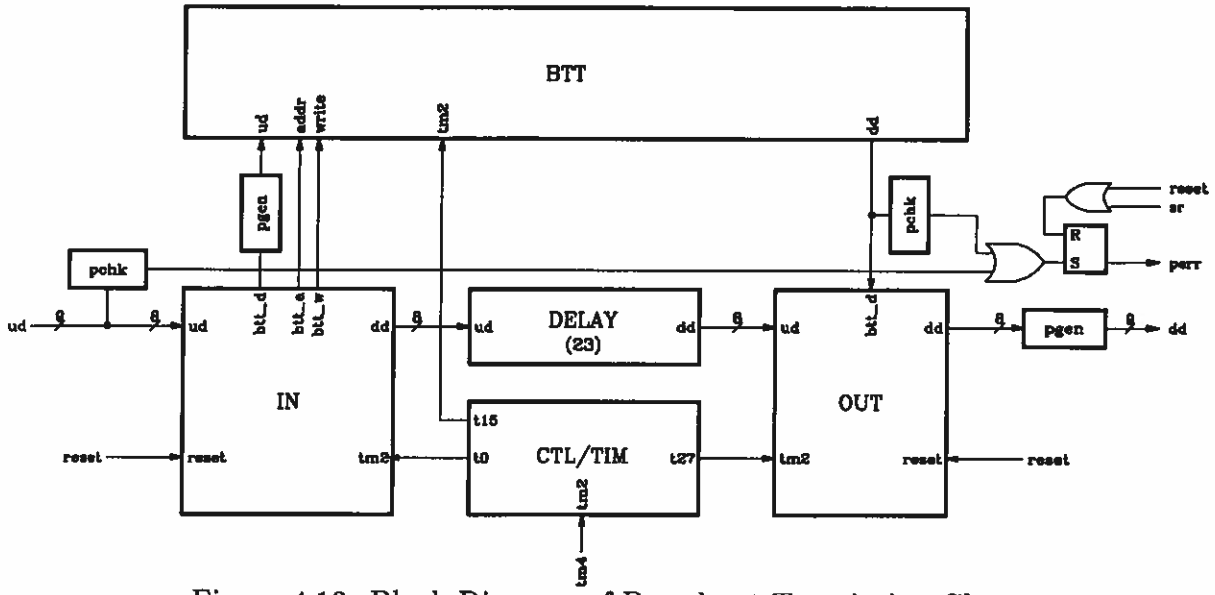

Figure 4.13: Block Diagram of Broadcast Translation Chip

The internal components that make up the BTC are similar to those in the PP. In particular, the BTC can be described as two SSPs along with a lookup table. This structure is illustrated in Figure 4.13. Specifications of the input and output SSPs have been written and the generated circuits have areas of about 4 mm^2 each.

5. Tools for Design of Communication Circuits

Faculty	Jonathan Turner
Research Associate	Pierre Costa
Graduate Student	Neil Barrett
	Gaurav Garg
	Tony Mazraani
	Anne Reynolds
	George Robbert
	Einir Valdimarsson

The implementation of the prototype packet switch for the ACS project requires several custom integrated circuits. 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 work on 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 developed a number of other tools to automate various 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.

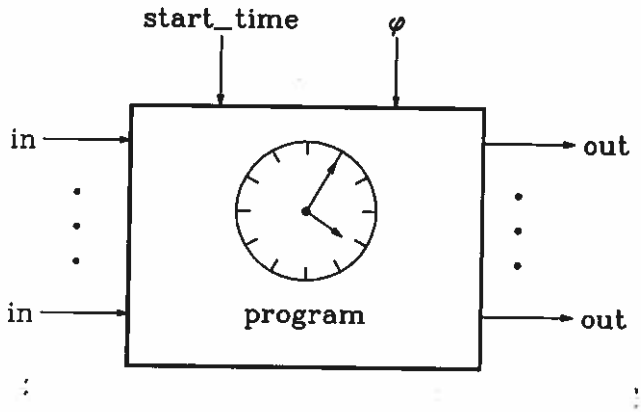

Figure 5.1: Generic Synchronous Stream Processor

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 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. 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. Our example is a 2×2 unbuffered switch element that could be used in a self-routing switching network. A block diagram of the switch element is shown in Figure 5.2. The headers of the incoming packets have a type field (`ptyp`) and an address field (`addr`). Each switch element uses the first (low order) bit of the address to select which port to output the packet on. The switch element also rotates the bits of the address to place the next address bit in the correct place for the next switch element to use in routing the packet. It may occur that both input

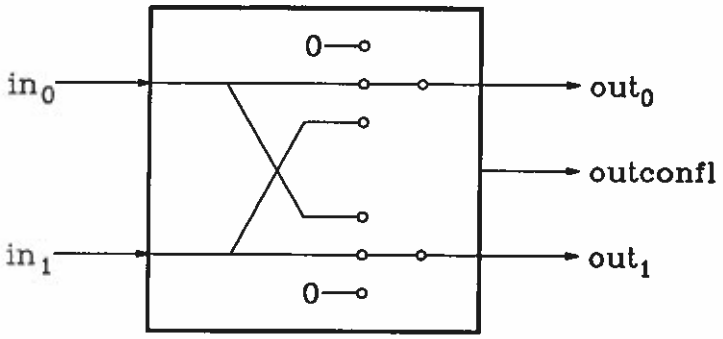

Figure 5.2: Unbuffered Switch Element

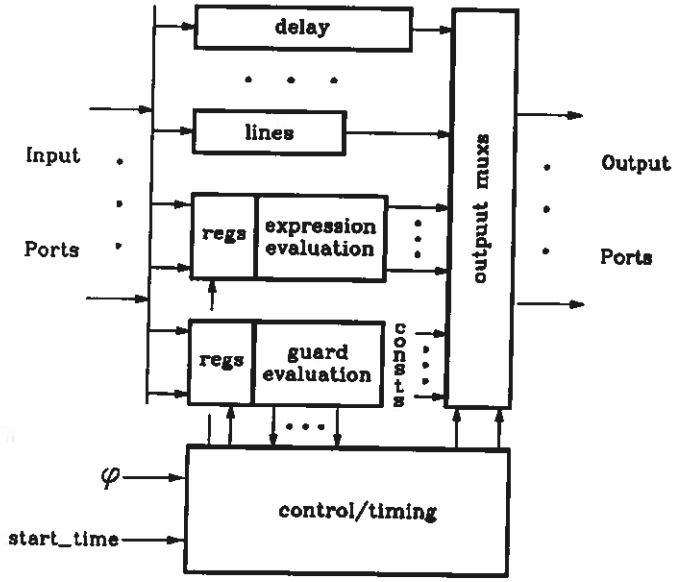

Figure 5.3: Target SSP Architecture

packets to the switch element request the same output port. If one of these does not contain useful data (`ptyp = PT_NONE`), no problem exists since that packet is dropped anyway. However, if both packets contain valid data (`ptyp = PT_PPOINT`), the "straight through" packets are given priority. That is, if both packets request output 0, the packet on input 0 is passed on and the packet on input 1 is discarded. Whenever a packet containing valid data is discarded, an error signal is asserted. The specification of this circuit is shown in Figure 5.5.

The initial `struct` declaration defines the `packet` format. The `#define` lines define constants and simple macros. The `Shipout` macro, when given the name

of an input and an output port, transfers a packet from the specified input to the specified output, while performing rotation of the address field. The first part of the `router` module definition specifies the module's input and output ports. `Port[8]` specifies that the following ports are eight bits wide. The symbols '<' and '>' specify the direction of the port with '<' for input and '>' for output. The number following the '@' specifies the time at which data starts to pass across the port. The `if` statement specifies five different conditions or *guards*. Notice that the guards are not mutually exclusive. When multiple guards are simultaneously satisfied, the actions associated with all of them are carried out. If two simultaneously true guards contain assignment statements giving a value to the same outgoing field, an arbitrary choice is made. 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 in the packet processor.

SSPs that perform simple functions, as are typical in the packet processors, fit nicely into a common architecture illustrated in Figure 5.3. 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.

We have developed 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 5.4. 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 PLA generator.

Figure 5.6 shows the intermediate specification produced by the compiler. The first few lines specify the name of the module, the module's ports including their names and widths and the maximum time that must be tracked by the module. The `condition` lines specify the conditions given in the guards of the `if` statement. The `outmux` lines specify the contents of each bit of each output port. The general form is

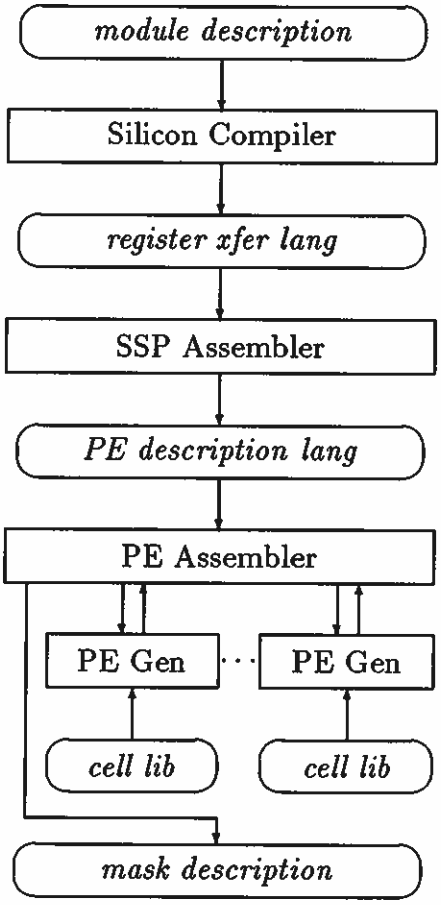

Figure 5.4: Structure of SSP Generator

`outmux` *output condition first_bit number_of_bits source*

When the specified condition is satisfied, the specified group of bits on the specified output port is obtained from the specified source. The source may either be a constant or the name of an input port along with a first bit on that input port. In these statements the bits are numbered serially, ignoring the port width (so bit 3 of the fourth word on a five bit wide port would be referred to as bit 22). So for example,

```
outmux _out0        1                    0      32          0
```

specifies that bits 0 through 31 of port `_out0` are assigned the value 0, while

```
typedef struct {
    bit ptyp[2];        /* packet type              */
    bit addr[6];        /* destination address      */
    bit data[72][8];    /* data in packet           */
}packet;

/* various packet types */
#define PT_NONE     0   /* no data in this packet  */
#define PT_PPOINT   1   /* point-to-point packet   */

#define Shipout(in,out) \
        out = in; \
        out.addr[5] = in.addr[0]; \
        out.addr:(0 .. 4) = in.addr:(1 .. 5);

router (port[8] packet
                <in0@0,  <in1@0,        /* input data ports  */
                >out0@4, >out1@4;       /* output data ports */
          port[1] bit >outconfl@4)      /* output conflict   */

{
    outconfl = 0;
    if (in0.addr[0] == 0 && in0.ptyp != PT_NONE) ->
        Shipout(in0,out0);
    |  (in1.addr[0] == 1 && in1.ptyp != PT_NONE) ->
        Shipout(in1,out1);
    |  (in0.addr[0] == 1 &&
              (in1.addr[0] != 1 || in1.ptyp == PT_NONE)) ->
        Shipout(in0,out1);
    |  (in1.addr[0] == 0 &&
              (in0.addr[0] != 0 || in0.ptyp == PT_NONE)) ->
        Shipout(in1,out0);
    |  (in0.ptyp != PT_NONE && in1.ptyp != PT_NONE &&
        ((in0.addr[0] == 0 && in1.addr[0] == 0) ||
         (in0.addr[0] == 1 && in1.addr[0] == 1))) ->
            outconfl = 1;
    fi;
}
```

Figure 5.5: Specification of Unbuffered Switch Element

```
name     router
output   _out1        8
output   _out0        8
input    _in1         8
input    _in0         8
output   _outconfl    1
maxtime 77

condition C0 (!_in0[2]&!(!_in0[1]&!_in0[0]))
condition C1 (_in1[2]&!(!_in1[1]&!_in1[0]))
condition C2 (_in0[2]&(!_in1[2]|(!_in1[1]&!_in1[0])))
condition C3 (!_in1[2]&(!!_in0[2]|(!_in0[1]&!_in0[0])))
condition C4 ((!(!_in0[1]&!_in0[0])&!(!_in1[1]&!_in1[0]))&
                  ((!_in0[2]&!_in1[2])|(_in0[2]&_in1[2])))

outmux _out0      1                    0     32        0
outmux _out0      1&!(C0|C3)          32    584        0
outmux _out0      C0&!(C3)            32      2     _in0:0
outmux _out0      C0&!(C3)            34      5     _in0:3
outmux _out0      C0&!(C3)            39      1     _in0:2
outmux _out0      C0&!(C3)            40    576     _in0:8
outmux _out0      C3                  32      2     _in1:0
outmux _out0      C3                  34      5     _in1:3
outmux _out0      C3                  39      1     _in1:2
outmux _out0      C3                  40    576     _in1:8
outmux _out1      1                    0     32        0
outmux _out1      1&!(C1|C2)          32    584        0
outmux _out1      C2                  32      2     _in0:0
outmux _out1      C2                  34      5     _in0:3
outmux _out1      C2                  39      1     _in0:2
outmux _out1      C2                  40    576     _in0:8
outmux _out1      C1&!(C2)            32      2     _in1:0
outmux _out1      C1&!(C2)            34      5     _in1:3
outmux _out1      C1&!(C2)            39      1     _in1:2
outmux _out1      C1&!(C2)            40    576     _in1:8
outmux _outconfl  1                    0      4        0
outmux _outconfl  1                    5     72        0
outmux _outconfl  1&!(C4)              4      1        0
outmux _outconfl  C4                   4      1        1
```

Figure 5.6: Intermediate Specification of Unbuffered Switch Element

```
outmux _out0      C0&!(C3)              34      5       _in0:3
```

specifies that when condition C0 is true and C3 is not, bits 34 through 38 are assigned the value of bits 3 through 7 on port _in0.

The assembler phase of the compiler produces a file that contains specifications of the input and output buses and processing elements required for the circuit. In the case of the router, this file specifies six bus lines on the input side in addition to those required by the input ports. These six lines include one for latching input signals and five for carrying the values of the guard conditions. The file also specifies 12 extra bus lines on the output side. These are used for gating various values onto the output buses. The specification of the guard evaluation logic requests a PLA with six inputs and five outputs. The delay specification requests a total of eight delay lines, each with an auxiliary output allowing the delayed value to be used for either of the two output ports. The delay line specification also includes specifications for several constants. The specification of the control and timing circuit requests 13 timing signals, one for latching input values and the remainder for controlling the timing and conditions under which values are gated onto the output bus.

The circuit generated from this specification contains about 2700 transistors and occupies an area of 3.7 mm^2 in two micron CMOS. Of this, roughly 59% is routing, 22% is the control and timing circuit, 16% is the delay lines and 3% is the guard evaluation logic. While the fixed floorplan results in circuits that are not terribly area efficient, the savings in design time more than compensates for the area usage which in typical applications represents a small part of the total area of chips in which SSPs are used.

The current version of the circuit generator *sspc* was written by George Robbert for his master's thesis. Since George graduated in June, the program has been ported to the Sun workstation environment, several bugs have been fixed and some simple optimizations added. It is now being used to generate circuits needed within the packet processor and broadcast translation circuit.

5.2. Memory Generators

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 5.7. Its external interface includes upstream and downstream 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

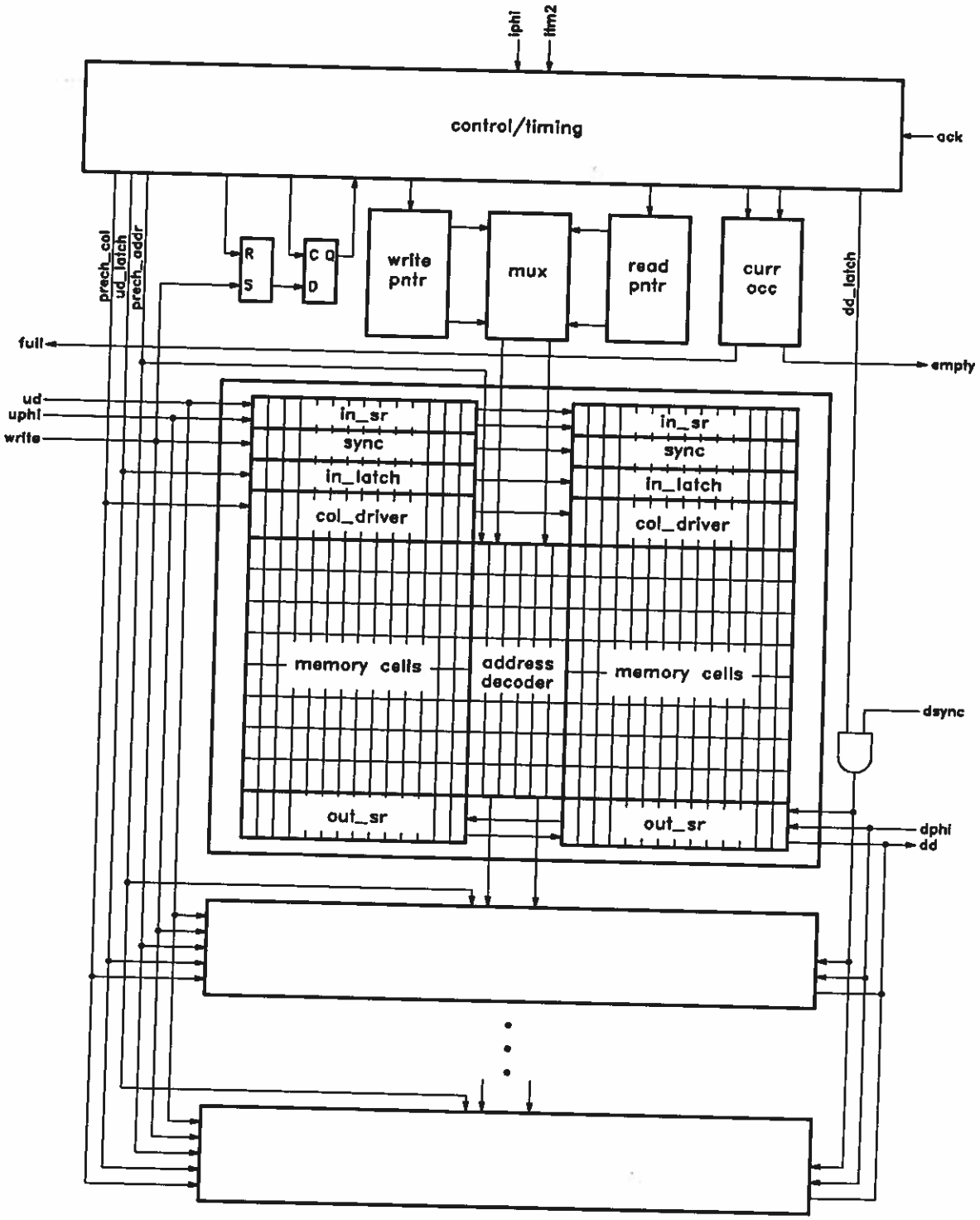

Figure 5.7: Packet Buffer

simulation studies indicated that the required refresh intervals might prove difficult to achieve. Incoming packets enter input shift registers at the top of each memory array and after the packet has been shifted in, it is transferred into the memory array. Outgoing packets are read from memory to an output shift register and then shifted out. The control section includes input and output pointers and a counter that keeps track of the number of packets in the buffer.

The packet buffer is designed to support different clock speeds on the input and output sides and asynchronous input. This is accomplished by having three distinct clocks. The internal clock determines the basic operating cycle of the buffer. During one of its internal cycles the buffer may be read once and written to once. The output clock may operate at the same rate as the input clock, or some integral divisor of the internal rate (typically half). During each output cycle the packet that is currently the first one in the buffer is read and shifted out on the downstream data lines (`dd`). If the downstream device acknowledges receipt of the packet at the appropriate time in the packet cycle, the control circuit updates the output pointer and packet counter appropriately. Note that while the output clock may operate at a different rate from the internal clock, the operation of the buffer on the output side is fully synchronous. On the input side however, asynchronous operation is allowed. In particular, a separate input clock is provided to control the input shift register and synchronization register. The write signal determines when data is transferred from the input shift register to the synchronization register. This signal can arrive at an arbitrary moment. The write signal sets an RS flip flop, which is latched at a fixed time in the buffer's internal cycle. The output of the synchronous latch is used only after it has ample time to settle. At this time, an incoming packet is transferred from the synchronization registers to an input latch. The only constraint placed on the input side when operated asynchronously, is that the input clock rate be no more than about half the internal clock rate. The input side may also be operated synchronously with the internal clock, in which case it may operate at the same rate as the internal clock.

We have written 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. The packet buffer generator is being used to construct all the buffers in the prototype packet processor. Integrated circuits containing large packet buffers for use in the XMB and RCB have been designed and submitted for fabrication.

A similar program has been written to generate lookup tables. A lookup table is a memory 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. At the start of each epoch the `address` and `write` signals are latched and data begins to enter. The table is read

```
condinps C0 C1 C2
muxouts o0 o1 o2 o3
regouts o4 o5
cntsize 7
cntmax 90
cntena --- 84<MAX>
sigset 001 30 o0
sigrst --- 32 o0
sigset --1 27 o1
sigrst -10 32 o1
sigset --1 38 o1
sigrst -10 45 o1
sigset ---  2 o2
sigrst -1- 70 o2
regclk --- 34 o4
regclk 111 56 o5
```

Figure 5.8: Control and Timing Example

or written on every cycle. On write cycles, data appears on the output ports at a specific time in the cycle. The program for the lookup table takes as parameters the number and size of the table entries, the data path widths and the target aspect ratio.

5.3. Control and Timing

All the circuits used in the prototype require control and timing circuits to synchronize various operations. A program to generate control and timing circuits was written initially for use in the synchronous streams processor generator, but we have found it useful in a variety of other contexts as well. The circuit generated provides two types of signals; *latch* signals are high during a single `phi1` pulse and *enable* signals are held high for multiple clock ticks. Enable signals change level while *phi2* is high. Both types of signals can be modified by a set of conditional inputs.

An example of a specification of a control and timing circuit appears in Figure 5.8. The `condinps` line lists the conditional inputs, the *muxouts* line lists the enable signals, and the *regouts* line lists the latch signals. The size of the timer register is seven bits and the maximum value that need be stored is 90. The `cntena` line specifies a condition and time at which counting should be disabled. In this

case, no condition has been specified, so the counter always stops at a value of 84. The remaining lines specify the various control signals. For example, the lines

```
sigset 001 30 o0
sigrst --- 32 o0
```

specify that the enable signal o0 is to be high at time 30 if C0 and C1 are both false and C2 is true, then reset at time 32 independently of the various conditions. The resulting pulse will be low during phi1 of time 32. Note that enable signals may be asserted several times during an epoch with different conditions.

The circuit generated by the program is a dynamic PLA with a counter to perform the timing function and a set of flip flops which control the enable signals. The circuit generated by the example above has an area of 0.3 mm^2 in 2 micron CMOS.

5.4. Test Vector Generation

One of the most time-consuming parts of designing an integrated circuit is simulating the circuit to verify its operation. This is particularly true for circuits like the packet switch element which must be simulated over several packet cycles in order to verify correct operation under the variety of conditions possible. Since each packet cycle (or epoch) is 86 clock ticks long and each tick of a two phase clock must be simulated by eight simulation steps, a five epoch simulation run comprises over 3,000 simulation steps. For each simulation step, roughly 30 input pins must be specified and another 30 or so monitored. Consequently, typical simulation input files are about 100K characters long and output files over twice that length.

Needless to say, generation of such large input files by manual methods is both time-consuming, tedious and error prone. Consequently, we have developed tools for automating much of this process. We briefly discuss one program that was developed for generating the test vectors for the packet switch element. A typical test specification for the packet switch element is shown in Figure 5.9. The first two assignment statements specify the operation mode (om) and stage number (sn) of the chip. Then, there is a sequence of assignments that specifies what is to happen on subsequent cycles. In these statements, udA and udB refer to the upstream data ports of the switch elements. Notice that the assignments specify the values in various fields of the incoming packets; rc stands for routing control, for example. The downstream information field (.i) is specified as a sequence of fields each containing three characters; the first two are interpreted as the contents

```
# operation mode = copy network , stage number = 1
        om = 3
        sn = 1

# packet cycle 1
        udA.rc = 2
        udA.op = 0
        udA.ln = 09
        udA.bcn = 1
        udA.i = aa1 551
        udB.rc = 1
        udB.op = 0
        udB.ln = 00
        udB.bcn = 0
        udB.i = f01 0f1
        dg0 = 1
        dg1 = 1
        go 1

# packet cycle 2
        udA.rc = 1
        udA.ln = 02
        udA.bcn = 0
        udA.i = 111 881
        udB.rc = 1
        udB.ln = 02
        udB.i = 661 991
        dg1 = 0
        dg0 = 0
        go 1

# packet cycle 3
        udA.rc = 4
        udA.ln = 00
        udA.i = ff1 ff1
        udB.rc = 0
        udB.ln = 00
        udB.i = 001
        dg0 = 1
        dg1 = 1
        go 1
```

Figure 5.9: Example Test Vector Specification

```
sn=1 om=3 dg0=1 dg1=1 ugA=1 ugB=1
    400 200 400 400
    091 001 040 051
    001 001 001 001
    010 001 010 010
    aa1 f01 aa1 aa1
    551 0f1 551 551
    aa1 f01 aa1 aa1
    .   .   .

sn=1 om=3 dg0=0 dg1=0 ugA=1 ugB=1
    200 200 001 001
    020 020 001 001
    001 001 001 001
    001 001 001 001
    111 661 001 001
    881 991 001 001
    111 661 001 001
    .   .   .

sn=1 om=3 dg0=1 dg1=1 ugA=1 ugB=0
    800 001 200 200
    001 001 020 001
    001 001 001 001
    001 001 001 001
    ff1 001 111 f01
    ff1 001 881 0f1
    ff1 001 111 f01
    .   .   .
```

Figure 5.10: Example Output

of an information byte in hexadecimal notation, the last is the incoming parity bit for that byte. This allows parity errors to be specified. If only a few fields are given, the information field is filled with repeated copies of the specified bytes. The downstream grants (dg) may also be specified, allowing one to specify tests which cause packets to be buffered in the switch element's internal buffers. The go statement causes the simulation to advance the specified number of epochs using the values specified above.

The program simulates the operation of the packet switch element at a functional level and produces three output files. One is a documentation file that specifies the input vectors and expected outputs in an easily understood form. A portion of the documentation file for the example is shown in Figure 5.10 The four columns specify the packets on the upstream data lines `udA`, `udB` and downstream

data lines dd0, dd1. The second file contains test vectors for the logic simulator, *esim* and the third contains the expected output from *esim*. For this example, both of these files are over 2000 lines and 100,000 characters long.

The program described was used to generate the vectors for the packet switch element, greatly simplifying the process and allowing more thorough testing than would otherwise have been possible. Similar programs have been developed for the packet buffer and lookup table. We plan to write still others for the PP1, PP2 and BTC chips. Our longer term goal is to develop more general methods that will allow automatic generation of test vectors for a wide class of circuits.

We have also developed another tool to enhance our existing logic simulator. The current simulator expects input data to be presented in a so-called horizontal form, where each input signal is one horizontal line in the input file. This is inconvenient for long simulation runs. We have written a program called VESIM that takes data organized in a vertical fashion, reformats it for use by the simulator then takes the resulting simulator ouput and formats it again into the vertical format. This facilitates long simulation runs comprising thousands of simulation steps. It also gives us a format that is fully compatible with the test equipment that we are using to verify operation of the actual chips.

5.5. Register Generator

A simple program called **mkreg** has been written to assemble commonly needed registers and counters. **Mkreg** can generate registers ranging from simple latches, to up-down counters with both serial and parallel inputs and high drive or tri-state outputs. The user specifies the required options and the number of bits and a file is created containing layout of the specified register. For example,

```
mkreg -n4 -o ex1.mag
```

specifies a simple four bit static latch and

```
mkreg -ISE -n4 -r12 -o ex2.mag
```

specifies an up-counter (I for increment), with parallel and serial loading (S for shift), tri-state outputs (E) and reset on a value of 12. This last option means that when the value of the register reaches 12, the next increment signal will give it a value of 1.

The registers are assembled from a collection of standard cells that have been designed to interface compatibly with one another. The program merely assembles

```
title = Sample Timing Diagram
xlabel = 1 unit = 5 ns
units 10
signal phi1
init value 1
start at 3
duty cycle .25
period 4
signal phi2
initial value 1
duty cycle .5
period 4
start at 0
signal sig1
init value 1
at time 2 level = 0
at time 3 level = 1
at time 6.5 level = dc
at time 9 level = 0
signal sig2
init value 0
at time .25 level = 1
at time 2 level = 0
at time 4 level = dc
at time 5.75 level = 1
at time 7.25 level = dc
at time 8 level = 0
at time 9.75 level = 1
signal sig3
init value 0
at time 0 level = 1
at time 2 level = 0
at time 9.25 level = 1
```

Figure 5.11: Example Input to Timing Diagram Generator

bit slices of the cells implementing the specified options, along with the requisite control circuitry. This program has been used to generate registers used in the packet switch element and packet buffers.

5.6. Timing Diagram Generator

We have developed a program to create timing diagrams from a text specification. The program creates a sequence of AutoCad commands that can be executed using

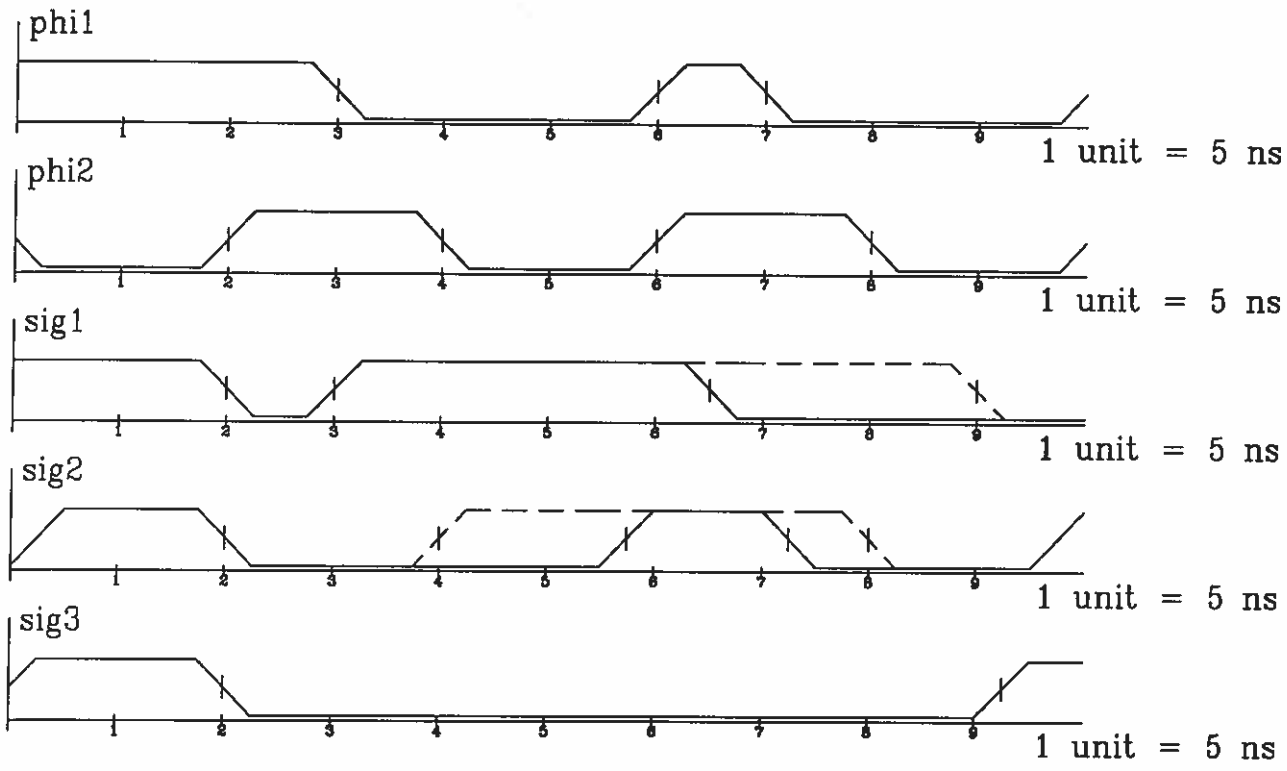

Figure 5.12: Generated Timing Diagram

the `script` command. Customization of the resulting diagrams is then possible within AutoCad. An example specification is shown in Figure 5.11 The `signal` lines introduce the specification of a new signal. Note that clock signals are easily specified and that signals may have don't care conditions. The timing diagram generated by this specification is shown in Figure 5.12.

6. Multipoint Routing

Faculty	Jonathan Turner
Research Associate	Makoto Imase
Graduate Student	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 cap: $E \rightarrow \Re^+$ and cost: $E \rightarrow \Re^+$ 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 [32]. This problem has been shown to be NP-complete by Karp [61] in 1972. Consequently one is forced to consider approximation algorithms which are not guaranteed to produce optimal solutions.

In previous reports we have described results of empirical comparisons of several known approximation algorithms for the Steiner tree problem, including the MST heuristic [52] and an algorithm due Rayward-Smith [76]. Our results indicate that both algorithms yield near-optimal solutions most of the time, with Rayward-Smith giving slightly better performance than MST. Until recently however, the worst-case performance of Rayward-Smith has remained open. We have recently proved that Rayward-Smith's algorithm has the same worst-case performance as MST. That is, it never produces solutions with cost more than twice optimal and

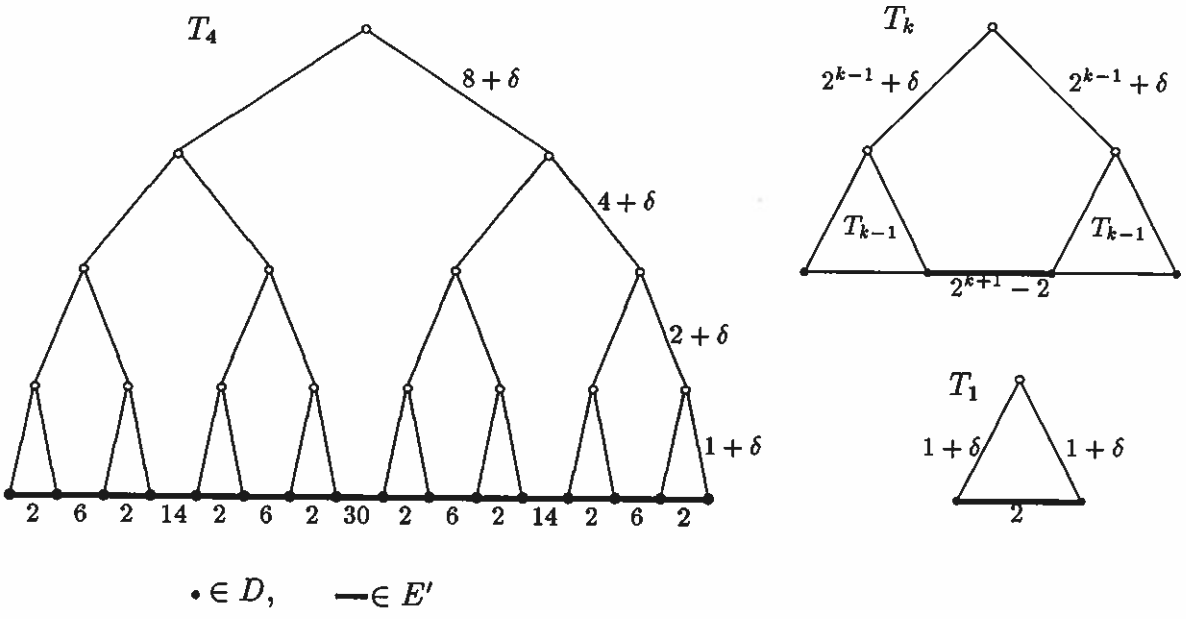

Figure 6.1: Worst-Case Example for Rayward-Smith's Algorithm

furthermore it can be off by as much as a factor of two. An example of a graph demonstrating the worst-case behavior of Rayward-Smith is shown on the left in Figure 6.1. In this example, the vertices to be connected appear at the bottom of the figure. Rayward-Smith's algorithm joins these vertices with the edges along the bottom, whereas the optimal solution uses the "tree edges." Note that the sum of the lengths of the bottom edges is almost twice that of the tree edges. The right hand side of the figure shows a general recursive construction for the worst-case example.

We are currently studying several algorithms which we believe may have worst-case performance better than two times optimal. One group of algorithms is based on the technique proposed by Rayward-Smith. These algorithms consider only those subtrees with more than two endpoints while contracting subtrees created at each stage of the algorithm to single nodes. Empirical data on one algorithm in which subtrees are restricted to three or more endpoints indicates that it yields a small improvement over Rayward-Smith's algorithm. A second group of algorithms uses MST to generate an initial solution and then in the second stage modifies that solution in an attempt to produce a solution of lower cost.

The classical approximation algorithms for the Steiner tree problem assume that the problem is static and can be solved in a centralized fashion. In a large

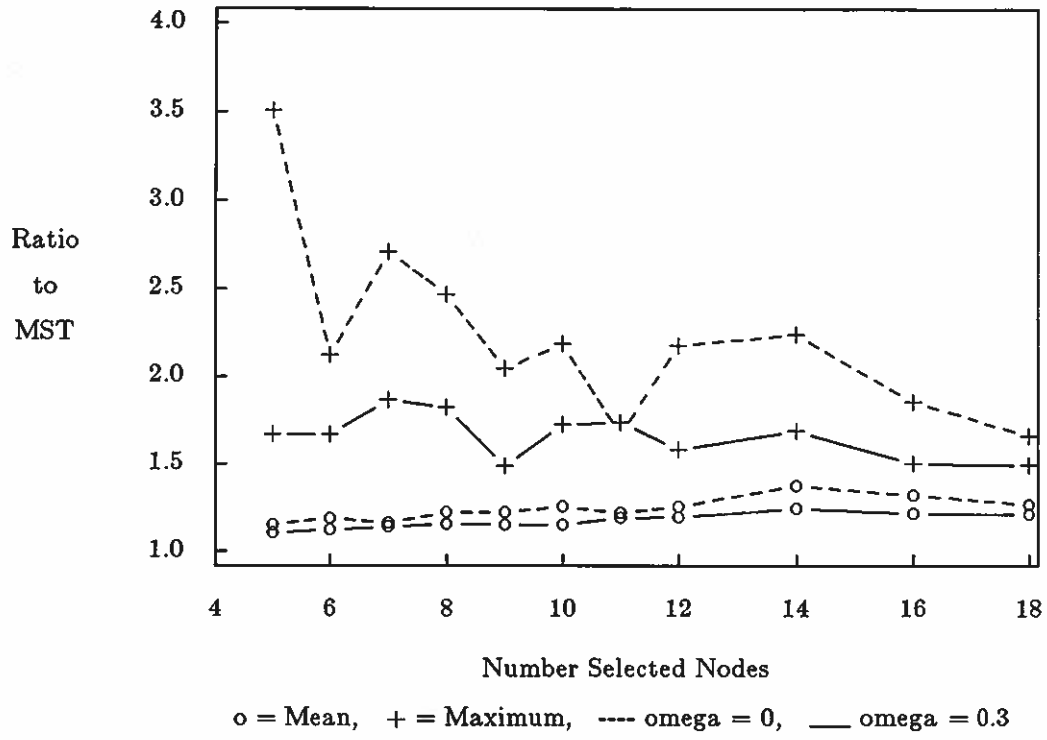

Figure 6.2: Empirical Performance of Weighted Greedy Algorithm

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 Rayward-Smith heuristics are most useful as tools against which the performance of more realistic algorithms can be measured. In previous reports we have described a simple greedy algorithm for the dynamic version of the multipoint routing problem and have reported on empirical studies of its performance. We have recently studied a weighted version of the greedy algorithm which has better average performance and is less susceptible to pathological behavior than the unweighted algorithm.

The basic 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. This algorithm can perform reasonably well, yielding solutions that are typically within about 30% of MST. On the other hand, its performance can degrade relative to MST when several endpoints drop out of a connection. This degradation is in part inherent in the fact that the algorithm simply prunes the connection when nodes drop out, but is exacerbated by the greedy way in which

endpoints are added in the first place. The weighted greedy algorithm attempts to improve the performance by making the choice of path to a new endpoint dependent not only on the length of the path, but also on the length of the path from the "junction point" to the "owner" of the connection. For our purposes, the owner o is just a distinguished endpoint which cannot be removed during the life of the connection. In the weighted algorithm, a node u is added to a connection by the shortest path from it to a node v, in the connection that 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 the 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. See Figure 6.2.

At the present time we are considering the worst case performance of several algorithms which may be implemented in a distributed fashion. For the greedy algorithm we know that the worst case performance is unbounded in the general case. In the case where we only add endpoints to the connection we have derived a bound of $\log k$ where k is the number of endpoints in the connection. We are also studying an algorithm for modifying an existing connection in a distributed fashion which we believe has a worst case performance within a factor of two times optimal.

In order to collect additional data on the performance of network routing algorithms we are now implementing a network simulator, which will generate multiple dynamic connections. A user will provide the simulator with a description of a network, a list of connection types, and a routing algorithm. The description of a network will include the network topology in the form of an undirected graph, with capacity and cost for each link. In addition a weighting factor for each node in the network will be used to indicate the relative frequency with which each node is included in a connection. The description of a connection type will include bandwidth, average duration, type (e.g. point to point or multipoint), average size, and relative frequency. Under a given load the simulator will yield information on carried load, blocking probabilities, and the costs of connections. Note that the definition of load is complicated by the fact that we must consider multipoint connections. We plan to use two definitions of load. In the first case we define the load induced by a single connection to be the bandwidth of the connection multiplied by one less than the number of endpoints. Under this definition the load for a point to point connection is just its bandwidth. In the second case we define the load induced by a connection to be the product of the bandwidth of the connection and the cost of an optimum route. Thus for point to point connections the cost will be the bandwidth of the connection multiplied by the cost of a short-

est path connecting its endpoints. For a multipoint connection the load will be the bandwidth multiplied by the cost of a minimum Steiner tree. Here we will make use of MST to approximate the load induced by a multipoint connection.

In our earlier work, we have studied the expected performance of various algorithms using empirical methods. In the last year, we have begun analyzing these algorithms in a probabilistic setting. Such studies require appropriate probability models and a detailed understanding of their behavior. The primary model for our studies is one in which nodes are distributed randomly on a unit square (or alternatively a unit sphere), and links are created between nodes probabilistically. The probability that two nodes are joined by an edge depends on the Euclidean distance between the two nodes, with nearby nodes having a higher probability of being joined than distant nodes. More specifically, for each pair of nodes u, v, an edge is created with probability

$$\beta e^{d(u,v)/\alpha L}$$

where $d(u,v)$ is the normalized Euclidean distance from u to v and α and β are parameters of the model which control the preference for short edges and the overall edge density. We have derived several basic results concerning the behavior of this and other related models. For example, for the spherical version of this model, we have derived the threshold function for connectivity.

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

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

We have also 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, Rayward-Smith's algorithm 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.

More 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 Bollobas [8] that if the edge probability is given by

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

then almost all graphs have diameter d for $C > 2$. If we let $k(n) = |D|$, where D is the set of nodes to be interconnected, we have shown that MST and RS will almost always produce solutions of identical cost when $k(n) < n^{(1-\epsilon)/d(d+1)}$, for $0 < \epsilon < 1$. In addition if

$$\epsilon n^{1/d(d+1)} < k(n) < n \ln)^{(1-\epsilon)/2d}$$

we have shown that Rayward-Smith's algorithm almost always produces a solution with cost less than that of MST.

7. Video Coding in Packet Networks

Faculty	Jerome R. Cox, Jr.
	Jonathan Turner
Research Associate	Pierre Costa
Graduate Student	Shabbir Khakoo

The use of packet networks for carrying video signals raises a variety of new research issues, which we have begun to investigate in the last year. We see three major sets of issues. The first set concerns the application of video coding to a diverse application environment. The second concerns the effect of packet transport on performance of coding algorithms and the design of protocols to limit undesired effects. The third concerns the implications of multicast communication for video coding. Each of these areas is reviewed briefly in this section.

7.1. Video Coding in a Diverse Application Environment

Currently, high compression coding techniques have been applied primarily for video conferencing, in order to reduce the bandwidth to the range of a few hundred Kb/s. While most of these techniques make use of motion compensation algorithms, they perform poorly in the presence of even moderate amounts of motion. We have investigated these algorithms, have determined the reasons for this poor performance and have devised a class of algorithms capable of far better performance in the presence of substantial amounts of motion. We believe that these algorithms will yield substantially better performance in a diverse application environment.

Most high compression video codecs in common use today are based on *transform coding.* First, an image is broken into sub-blocks of (typically) 16 by 16

pixels and then for each sub-block, a two-dimensional transform such as the 2D discrete cosine transform (DCT) is computed giving a 16 by 16 matrix of transform coefficients. The transform coefficients form an alternative representation of the initial sub-block, which has the advantage that the perceptually important information is concentrated in a relatively small number of transform coefficients. Consequently, one can transmit a fraction of the transmit coefficients (or transmit them with varying precision) and recreate the image from those few coefficients without significantly degrading the image.

If one is transmitting a sequence of similar images, as in a video sequence, additional compression can be obtained by maintaining a copy of the previous image and transmitting the difference between the current image and the previous one. Typically, each sub-block to be transmitted is compared to the corresponding sub-block in the reference image, a difference sub-block is calculated and the DCT of the difference sub-block is computed and sent with varying precision. This can be improved further through a form of motion compensation called *block matching.* This involves comparing the sub-block not just to the corresponding sub-block in the reference image but to all sub-blocks of the reference image within a given distance of the sub-block to be transmitted. The coder then identifies the sub-block of the reference image that differs least from the given sub-block and transmits the difference information relative to that sub-block, along with the identity of the selected reference sub-block. See [65,66] for additional details.

A key element in the performance of a block matching codec is the algorithm used to identify the sub-block of the reference frame that is most similar to the current sub-block. To compare two sub-blocks, one typically computes the sum of the squares of the differences between corresponding pixels. For 16 by 16 sub-blocks this requires 256 multiplies and 512 additions. To compare a given sub-block to every reference sub-block within say eight pixels in any direction, requires 289 comparisons or a total of about 220,000 arithmetic operations. Since in a 512 by 512 pixel image, this must be done 1024 times per frame, the computational requirements are pretty clearly prohibitive for real-time coding.

Consequently video codecs that employ block matching don't attempt to consider every sub-block in the region. Rather they try to find the best match using some form of local search. Such techniques can work well if there are no local minima in which the search algorithm can get stuck. We have found however that for many typical images, local minima are common, and that the local minima are typically not nearly as good as the global minimum. Consequently, conventional search techniques often fail to achieve the highest possible compression rates.

We have devised a class of search algorithms that attempts to solve this problem. It is based on the idea of computing a concise *signature* for each sub-block

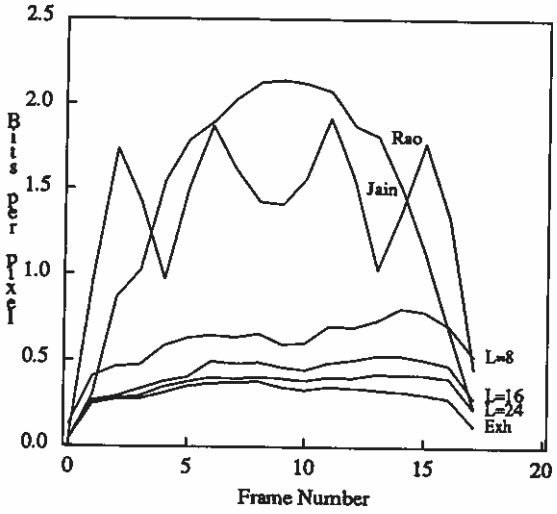

Figure 7.1: Compression Rate Comparison

in the search region and then comparing the input sub-block to the reference sub-blocks on the basis of their signatures. The signatures must be quickly computable and small enough that it's reasonable to perform an exhaustive comparison of the signatures. Then several sub-blocks with closely matching signatures are compared to the input sub-block using the squared-difference measure and the best match selected. We have evaluated one version of this technique in which the signatures are selected coefficients of a one-dimensional DCT, computed over the sum of the pixel values in each row of a sub-block. Figure 7.1 shows the results of one of our experiments. This plot shows the average number of bits per pixel for a sequence of frames starting from no motion, increasing to a panning motion of eight pixels between successive frames and then decreasing to no motion. The curves labelled Rao and Jain give the performance for two popular algorithms that are commonly used in video conferencing codecs [44,83]. The curve labelled exhaustive gives the performance obtained by exhaustively searching for the best match and the three other curves give the performance of three variations of our signature-based algorithm. The parameter N refers to the number of sub-blocks with closely matching signatures that were selected for the final comparison. Notice that the signature-based algorithm gives compression rates of three to five times that achieved by the

other algorithms in the presence of substantial amounts of motion. More detailed results may be found in [50].

The signature-based search algorithms promise far better performance in the presence of motion than conventional algorithms. This should allow them to operate successfully in a diverse application environment including video sources with a wide variety of different statistical characteristics. Another important issue in such an environment is the need for a variety of video formats. In a ubiquitous public network, it would be desirable, for example, to support video signals with a range of different resolutions, in order to allow individuals to purchase displays that suit their needs and budget. An equally important consideration is that technological improvements will continue to make higher resolution displays more affordable and as this happens, it should be possible to incorporate them into the network smoothly with a minimum of disruption. Finally, video will be applied increasingly to computer-based applications in which the signal is sent to a window on a computer workstation. The size and resolution of this window should be variable, so that the user can adjust them to suit his needs.

These considerations suggest the need for a video coding standard that works well across a wide range of resolutions. Such a standard would specify the signal resolution (and possibly other parameters) as a variable whose value is specified only as part of the actual signal. This would allow a transmitter to change its resolution as appropriate to a given situation. The coding standard should be designed to permit economical implementation of decoders that take a signal at an arbitrary resolution and display it at the available display resolution. This requires that information at different resolutions be easily separable, a property that of course is inherent to popular transform coding techniques. Given such a coding standard, a signal from a given source can be broadcast to a collection of receivers which can each display the signal at the local display resolution. Furthermore, if the fine resolution information is transmitted in separate channels, the network can deliver it only where needed.

It may also be useful to consider varying *temporal resolution* in addition to varying spatial resolution. This is already used in video conferencing codecs which typically transmit 7.5 video frames per second as opposed to the standard 30 frames. Security applications are an example of situations where even slower rates would be useful. Faster rates may also be important in certain high performance applications. While there are no great conceptual difficulties to providing variable frame rates, a standard format that supports it is needed to make it viable.

7.2. Packet Transport of Video Signals

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. 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 [65] 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 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. This is particularly important for applications involving scene changes, where it is necessary to transmit a completely new image in short period of time. Given the ability to transmit at high rates during rapid motion, codecs operating in a packet environment can be designed to minimize the average data rate rather than the peak. This is a significant advantage, since studies of the statistical properties of video signals show that even signals characterized by lots of motion, are very asymmetric in the sense that the high motion segments constitute a fraction of the total [46]. At the same time, the data rate variations can't be completely unconstrained as the degree of burstiness does have an impact on the amount of statistical bandwidth sharing possible in the network. While our work on buffer and bandwidth management provides some insight into how bursty sources affect queueing in the network, more detailed studies of the statistics of video sources are needed in order to to develop an understanding of the interactions between coding methods and network performance.

Packet transport of video signals also raises some new concerns, since lost packets can degrade the quality of the received signal. Coding algorithms that exploit temporal redundancy are especially sensitive to packet loss, since it can lead to inconsistent reference data between transmitter and receiver and if uncorrected, to gradual degradation of the image. As an example, for a standard resolution signal with 16:1 compression, one lost packet with 4096 data bits affects 32 of 1024 sub-blocks. Consequently, the effect of the lost packet can be significant. On the other hand, if the loss rate is low enough, packets are rarely lost. For example, for

a loss rate of 10^{-6}, one packet is lost about every 15 minutes. Of course, this is just a single example. In general, increasing the resolution of the signal decreases the impact of a lost packet on the image but increases the frequency. Increasing the compression ratio has the opposite effect.

There are several options for reducing the impact of packet loss. One possibility is to use interlaced coding and transmission. That is, the video signal is organized as two interlaced fields and these fields are separately coded and transmitted, allowing lost information to be recovered by interpolation from the other field. Another useful technique is background updating; during each frame period a certain fraction of the sub-blocks can be transmitted in uncoded form, allowing inconsistencies between transmitter and receiver to be corrected. If for example, one sub-block in 64 is transmitted without coding in each frame period, the entire image would be sent roughly every two seconds, while the extra overhead is small enough to be negligible. If the loss rate is low enough, on-demand retransmission may be viable; the objective would be primarily to maintain long-term consistency between the transmitting and receiving frame buffers. Another variation on the background updating scheme is multi-resolution updating. In this scheme, a low resolution version of the image is sent without coding on a fairly rapid basis (perhaps two to four times a second), while higher resolution information is filled in on a slower time scale. This technique is similar to progressive transmission of still images [25].

7.3. Implications of Multicast Environment

The multicast environment provided by a broadcast packet switching network creates several new issues relating to video coding. One such issue was mentioned earlier. When a video signal is being sent over a multicast connection to receivers with differing display resolutions, it's possible for the network to optimize bandwidth use by delivering the high resolution portion of the signal only to those endpoints that require it. This requires only that the high resolution information be carried in a separate sub-channel of the multipoint connection; given this separation, it is a simple matter for the network to provide it only where needed.

Another situation where the multicast environment raises new issues is in applications with many video sources that are to be combined to produce a display. One example is a teleconference with several participants at different locations. If simultaneous display of all the participants is desired, we have the issue of how to format and transmit the information without excessively loading the internal links of a multipoint connection. One possibility is to have each participant's codec transmit at the ultimate display resolution rather than the source resolution. With this approach, the network need only provide sufficient bandwidth for

one source rather than all sources. Another option is to transmit high and low resolution information in packets of differing priority. The network would deliver all the traffic if enough bandwidth is available but would preferentially discard low priority information in the event of congestion. If several priority levels are used, the network can adaptively accommodate changes in the number of transmitters, making it unnecessary for the transmitters to explicitly change their coding rate in response to the number of participants in the conference.

Another issue that arises in a multicast environment is the problem of cutting into a video signal that is already in progress. There are many applications where it will be important to allow receivers to join a multicast connection at arbitrary times. However, if coding algorithms that exploit temporal redundancy are used, a receiver joining a connection will lack the reference information needed to properly interpret the coded video stream. This is essentially a more severe version of the problem of packet loss. It is more severe because it can happen frequently and because all of the reference information is missing, rather than just a small part. One solution is, low speed transmission of the uncoded image. To achieve acceptable subjective quality, this is likely to require transmission of the high resolution information at a higher rate than the low resolution information. As an example, if we transmit a 512 by 512 pixel signal four times per second at one eighth resolution, once per second at one fourth resolution, every four seconds at half resolution and every eight seconds at full resolution, we use less than 1 Mb/s for this background information. This allows a receiver breaking into an ongoing connection to produce a subjectively satisfactory image within a second or two.

8. High Speed Internetworks

Faculty	Guru Parulkar
	Jonathan Turner
Graduate Student	James Sterbenz

In this section we address the issue of high speed communication in an environment comprising a mix of subnetworks with widely varying characteristics. The work of the ACS project and other research organizations is expected to lead to the development of large public networks capable of supporting applications ranging from low speed data to voice, high speed data and video. If such networks are to realize their full potential, they must be designed to operate in an environment that includes networks with widely varying characteristics.

Since the early seventies, much of the work on computer communication has been directed toward the development of protocols that allow interworking among computers, operating systems and communication subnetworks of different types. These efforts have culminated in the ARPA Internet Protocol Suite which has introduced a number of ideas of fundamental importance. Since the development of the internet protocols, the technological context in which we find ourselves has changed dramatically. The development of high speed LANs and workstations, and the growing role of supercomputers in scientific computing have led to new and largely unfulfilled requirements for high speed computer communication. These needs have been difficult to satisfy for a combination of reasons. First, existing wide-area computer networks have been unable to support the data rates required and second, the existing host computers are unable to deliver the data to the application at those rates.

On the other hand, fiber optic transmission systems are being introduced rapidly into the national communications infrastructure offering vast amounts of bandwidth at fairly modest costs. Several research groups at industrial and academic laboratories around the world have demonstrated that new high speed packet switching techniques can make these resources available in a flexible fashion, but

up to now these groups have failed to consider the need to operate in a complex networking environment consisting of autonomous and/or technologically dissimilar subnetworks. We feel that it is important to recognize that this kind of heterogeneous environment is here to stay and if we are to make the best possible use of new developments in networking, we need to establish a framework that supports such diversity.

8.1. Elements of an Extended Internet Model

The ideas developed in the ARPA Internet protocols are important ones, because they demonstrate that it is possible to build systems that support interworking across independently administered and technologically dissimilar subnetworks. We believe that the existence of such subnetworks is a fact of life and that continuing technological change and organizational imperatives will ensure their future proliferation.

Given then that future networks will operate in a heterogeneous environment, how must we extend current internet concepts to take advantage of the high speed communication technologies now under development? We feel that the internet model must be extended in three major ways. First it must support a connection-oriented transport service at the internet level that can support applications with demanding performance requirements. Second it must support a more general addressing scheme, to allow interworking among diverse subnets. And third, it should provide a framework for parametric description of subnet capabilities and connection requirements, allowing the routing of connections through subnets with appropriate capabilities in an application-independent fashion. These elements are discussed briefly below.

A Connection-Oriented Transport Service

One common element in most work on high speed networks is the use of connection-oriented packet switching. There are several reasons for this. Perhaps the most obvious is performance. Connection-oriented systems separate the more complex control operations from data transfer, allowing simple and fast hardware implementations of the data transfer. This is an important issue both within switching systems and within the various devices that communicate across the network. We feel that this separation of control and data will also be important in delivering high performance to computer applications, since it will facilitate the development of hardware-based mechanisms for speeding data transfers into and out of computer systems.

A second reason that connection-oriented networks are attractive is that they allow the network to make explicit resource allocation decisions when connections are established, and this in turn makes it possible to offer far more predictable performance than is possible in connectionless networks. Such predictable performance, while not always essential, is necessary for applications like interactive voice and video. Such resource allocation decisions must be made using some form of rate specification that describes the amount of bandwidth the user requires. For applications that don't have hard requirements, one can have "soft specifications" that give a range of acceptable possibilities, and allow the network to make its resource allocation decisions relative to other traffic. This can be extended further; we can include connections with "degradable specifications," and allow the network to take resources away from such connections in order to accommodate new traffic.

Finally, connection-oriented networks offer more generally useful methods of multipoint communication than is possible in truly connectionless networks. Connectionless networks can support multipoint communication in one of two ways; transmission of packets with a list of destination addresses or by transmission to a multicast address. The first mechanism is useful only for multicasts with a small number of endpoints and the second is essentially a form of connection, since it must be possible to associate a given set of endpoints with a given multicast address and these associations can be expected to change over time.

The arguments given above imply that an extended internet model should include a connection-oriented service. We refer to the connection-oriented service as ICP for internet connection protocol. This would not replace the existing connectionless service but would likely be the method of choice for applications that are connection-oriented at a higher level. ICP would support connections with resource requirements described by a general rate specification and would support multipoint connections. It would not provide perfectly reliable connections; that is it could deliver packets out of sequence or not at all. It would be designed to support very high speed connections and in particular would be structured to allow data transfers to be handled by simple hardware.

A Flexible Internet Addressing Scheme

One of the key requirements for an internet that can support communication across diverse subnetworks is a flexible addressing scheme that allows one to specify terminal devices that may be located in any of the subnetworks. Some of the specific objectives we have in mind are listed below.

- We want a scheme that allows us to address devices on existing subnetworks as diverse as current telephone networks (both public and private),

the ARPAnet, X.25 data networks and local area networks. Note that these networks currently use completely different forms of addressing and it is unrealistic to expect them to change.

- The addressing scheme should support a hierarchical organization of the address space, allowing the separate entities responsible for various subnetworks to manage their portions of the address space independently of the others.
- In order to support very large networks the addressing method must support routing methods that are not dependent on detailed global knowledge of network topology or traffic.
- The addressing method should allow for devices that have an identity independent of their current network location, facilitating the relocation of terminal devices and communication among mobile devices that may not have any fixed location.

None of the addressing schemes currently in use provides the level of flexibility needed in a general internet environment. In particular, none allows for operation in an environment that includes subnetworks with their own addressing schemes that may be very different from one another. We next outline a framework for addressing that would allow co-existence of such diverse subnetworks. Our framework is driven be our desire to support interoperation among existing and future subnetworks, without imposing the unrealistic requirement that their native addressing methods be abandoned. Our framework supports a hierarchical organization that allows routing to be carried out without the need for detailed global information. We envision this hierarchy to be based on routing knowledge rather than on physical connectivity. The proposed form of an address in our framework is as follows:

$$\text{OBJ} :: \text{DOM}_1 :: \text{DOM}_2 :: \text{DOM}_3 :: \ldots$$

OBJ denotes an *object*, which can designate a physical terminal device or some other entity within a terminal. Object identities must be unique across the entire network, a requirement that can be fulfilled by having the identity incorporated into a device at the time of its manufacture. DOM denotes a *domain*, which is either a subnetwork or a subnetwork together with the address of some entity within the subnetwork, where such an internal entity is identified using the subnetwork's native addressing scheme. Subnetwork identities are globally unique, a requirement which implies the existence of an organization or organizations to assign them.

To be useful, the particular sequence of domains in an address must tell us something about routing. We require only that DOM_i have a mechanism for determining how to reach DOM_{i-1}. This allows (but does not require that) the "path"

implicit in the address to be used for routing. In general, when a route is required for a given address, local routing information would be used at various switching systems and gateways to find a short path. So for example, if a path to the device specified by OBJ were known by a particular switching system, that path could be used, ignoring the remainder of the address. Similarly, if a given gateway knew how to reach DOM_2, it would not be obliged to go through DOM_3. Note that there is no requirement that DOM_i be physically connected to DOM_{i-1}, only that it be able, using whatever internal mechanisms it has at its disposal, to find DOM_{i-1}.

The OBJ portion of the address is optional, allowing for anonymous objects identified only by their location. This is needed for interworking with networks that support only anonymous devices (such as telephone networks), and relieves simple devices of the requirement for explicit identities. As indicated above, DOM_i may specify either a subnetwork or an entity within a subnetwork. The appropriate form would be determined by the subnetwork. For example in a subnetwork in which every switch/gateway knows how to reach every other, DOM_i could specify just the subnet, rather than a particular entity on it. On the other hand, in a large wide area network where individual switches/gateways have only partial knowledge, DOM_i might specify both the subnetwork and the identity of a gateway that joins the subnetworks of DOM_i and DOM_{i-1}. Alternatively, DOM_i might give the identity of a routing processor within the subnetwork. The case of the first domain, DOM_1 is a little special; in particular, when no explicit object identity is provided, DOM_1 must include the address within the subnet of the object.

This kind of a framework allows for the interoperation of existing networks, including some very dissimilar ones. For example it allows a telephone call to be placed from an appropriately equipped personal computer located on a local area network, to an ordinary telephone on the existing telephone network. It also allows connections between LANs to be routed using switched connections in the telephone network, if appropriate. It supports flexible routing for private wide area networks, allowing for example, that routes enter the private network at the most convenient gateway, rather than requiring (as with current private wide area telephone networks) that they enter at a particular place. This in turn allows transparent movement of devices within such subnetworks.

Parametric Description of Subnet Capabilities

Given the variety of capabilities of the subnetworks included in an extended internet, it is essential that the internet protocol include mechanisms for describing the capabilities of subnetworks, so that routing decisions can be guided by this information. For example, when selecting a route for a connection requiring a

bandwidth of 1 Mb/s it is essential that the route not traverse subnetworks incapable of supporting that bandwidth. Similarly connections requiring low packet loss rates should not be routed through subnets that lose packets frequently.

The following list gives a few of the parameters that might be included as part of a subnet description. A few of these parameters are given relative to a "standard reference path," which for example might be a path carrying heavy traffic between devices that are say 1000 km apart.

- *Bandwidth options.* This specifies the various connection bandwidths that the subnet can support. It may be specified as a few discrete values or ranges of values.
- *Bandwidth allocation option.* This specifies the type of bandwidth allocation that the subnet can support. Options include peak bandwidth allocation, in which bandwidth is dedicated to a connection and cannot be statistically shared with other connections; statistical allocation, in which connections with varying instantaneous data rates can statistically share bandwidth, but with explicit allocation provided to ensure predictable performance; and no bandwidth allocation, in which no performance guarantees are provided.
- *Packet loss rates.* This specifies the frequency of packet loss on a standard reference path.
- *Packet misordering separation.* This specifies the time between transmission of packets on a standard reference path at which the likelihood of packet misordering exceeds some threshold (say 10^{-3}).
- *Packet delay.* Specifies delay on a standard reference path (perhaps average and ninety-ninth percentile).
- *Multipoint capability.* The ability to support multipoint connections.
- *Transit traffic.* This specifies a subnet's willingness to carry transit traffic, that is traffic that crosses the subnet but does not terminate at some device on the subnet.

The parameters listed above are envisioned as static. It may also be useful to allow more dynamic traffic information to be included and updated periodically. Using these parameters and possibly others together with knowledge of individual connection requirements, it is possible for the ICP protocol entities to make informed decisions when routing connections.

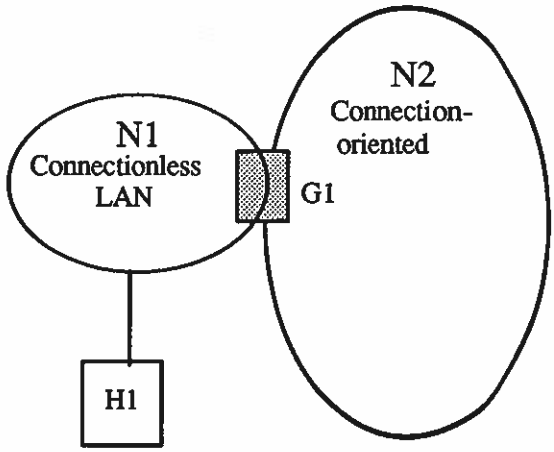

Figure 8.1: Connection Across a LAN

8.2. Design and Implementation Issues

In the previous section we have presented a number of ideas that we believe should be included in an extended internet model. We have also begun to explore some of the resulting design and implementation issues. In particular, we have considered the issue of how one supports the ICP protocol across a connectionless subnet, the converse problem of how one supports a connectionless service across a high speed connection-oriented subnetwork and the issue of how the host architecture can be modified to exploit the separation of data transfer from control provided by the ICP protocol so as to achieve very high performance.

Consider Figure 8.1. How should we implement a connection across the LAN which has no concept of connection? There are two major issues involved. First, how is a connection established and second, once it is established, how are packets transferred from the gateway to the appropriate host and vice-versa. One possibility is to include the connection establishment functions within the gateway; that is hosts requiring connections would exchange control messages with the gateway at the time of connection establishment, and the gateway would in turn exchange control messages with the connection-oriented network (N2) to complete the connection. Assuming the connection can be established, the gateway and H1 must agree on a way to identify packets belonging to the new connection. Then during the data transfer phase, the main gateway function would be to perform a routing translation; that is when a packet is received from N2 on the logical channel selected during connection establishment, it would be reformatted to include the address of H1 and whatever local identifier agreed upon by H1 and the gateway; in the other direction a similar translation would be performed.

One of the gateway's functions would be to monitor the usage of resources on the access link to N2, rejecting new connections that would overload it. In addition, the gateway should ideally monitor the LAN traffic and take it into account when accepting new connections. The monitoring function is not difficult to implement if one takes advantage of the broadcast nature of most LANs. On the other hand since much of the traffic on the LAN is not under the control of the gateway, resources available at connection establishment may not continue to be available later. How serious this issue is clearly depends on the proportion of the traffic that is subject to explicit resource allocation.

We note that the connection establishment functions need not be built into the gateway device. Any host on the LAN could implement these functions, allowing the gateway to be implemented as a relatively simple and fast hardware device. It's functions would in this case be reduced to packet relaying, using information in control tables that can be updated by the controlling host. This example illustrates some of the issues associated with operating connection-oriented and connectionless internet protocols within an environment comprising subnetworks with limited capabilities. Other cases are considered in [68]

We now briefly discuss some of the host interface issues. A typical networked host, is structured with a network interface sitting on the host's primary bus and effecting communication by transferring information to and from memory under control of the processor. When an incoming packet is transferred to memory, the processor is interrupted and subsequent processing typically involves a few decapsulations of the packet, scheduling of a few processes, processing of a few interrupts, and maybe copying the packet a couple of times[13]. All this processing essentially translates to throughput which is much lower than the raw data rate. A number of optimizations have been suggested to make these steps efficient within the host's operating systems in order to achieve higher throughput [14,15], but these must be viewed as stop-gap measures that cannot be expected to provide the order of magnitude improvements needed to take advantage of the high speed networks becoming available.

We propose a modified host architecture shown in Figure 8.2. Important aspects of this model are summarized below:

- The model includes a separate network bus which is used for getting packets into and out of the host, and thus, relieves the processor bus of this traffic.
- We propose that there be several transport level protocols optimized for different classes of applications. These protocols can be part of the network bus interface of various devices connected to the network bus, and thus can be optimized to the applications using those devices. For example, a transport protocol for the frame buffer can be optimized for video applications.

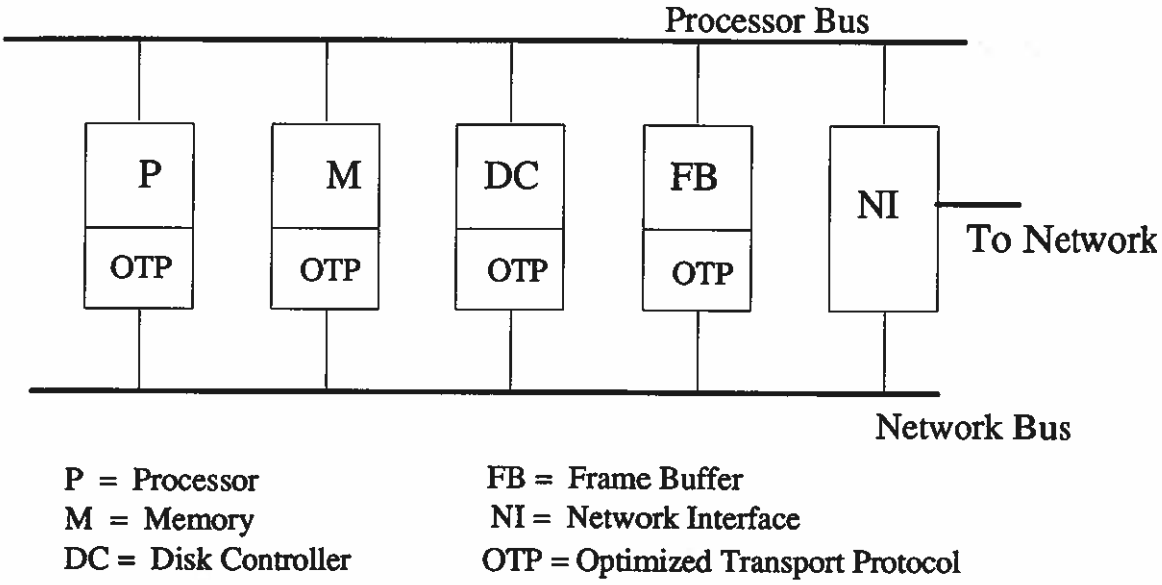

Figure 8.2: Proposed Host Interface Architecture

- Depending on application, the interaction between a user process running on the processor and the transport protocol can be such that most of the per-packet processing is done by the transport protocol on the interface in hardware and data copying is avoided by allowing the interface to deliver data directly to the application.

  For example, in a high speed file transfer protocol, application level entities would initially negotiate name, type, and place of the file with options and pass this information to the transport protocol. But during an actual data transfer, the transport entities would retrieve and store the data as fast as possible without any intervention from the software. The blocks that are received with error or not received at all will be noted but retransmission will be done only at the end in a second round, thus not throttling the sender and the receiver with the error control mechanisms. (Some of these issues are discussed in more detail for a similar environment in [15].)

We note that our model also includes the current approach of delivering data to memory where further action is taken by software processes. This option would be exercised for example where the data requires additional processing or where the need for flexibility is more important than the need for high performance.

Bibliography

[1] Agrawal, D. P. "Testing and Fault Tolerance of Multistage Interconnection Networks," *Computer*, 4/82.

[2] Akhtar, Shahid. "Congestion Control in Fast Packet Networks," Washington University Electrical Engineering Department, MS thesis, November 1987.

[3] Bassalygo, L. A. "Asymptotically Optimal Switching Circuits", *Problems of Information Transmission*, Vol.17, 1981, pp. 206–211.

[4] Batcher, K. E. "Sorting Networks and Their Applications," *Proceedings of the Spring Joint Computer Conference*, 1968, 307–314.

[5] Beckner, M. W., T. T. Lee, S. E. Minzer. "A Protocol and Prototype for Broadband Subscriber Access to ISDNs," *International Switching Symposium*, 3/87.

[6] Benes, V. E. *"Mathematical Theory of Connecting Networks and Telephone Traffic,"* Academic Press, New York, 1965.

[7] Benes, V. E. "Blocking States in Connecting Networks Made of Square Switches Arranged in Stages", *Bell Syst. Tech. J.*, Vol.60, Apr.1981, pp. 511–521.

[8] B. Bollobas. "The Diameter of Random Graphs," *Transactions of the American Mathematical Society*, 267:41–52, 9/81.

[9] Bondy, J. A. and U. S. R. Murty. *Graph Theory with Applications*, North Holland, New York, 1976.

[10] Bubenik, Richard. "Performance Evaluation of a Broadcast Packet Switch," Washington University Computer Science Department, MS thesis, 8/85.

[11] Bubenik, Richard and Jonathan S. Turner. "Performance of a Broadcast Packet Switch." Washington University Computer Science Department, WUCS-86-10, 6/3/86.

[12] Cantor, D. G. "On Non-Blocking Switching Networks", *Networks*, Vol.1, No.4, Winter 1971, pp. 367–377.

[13] Clark, D. D. "Modularity and Efficiency in Protocol Implementation," *ARPA RFC 817*

[14] Clark, D.D. "The Structuring of Systems Using Upcalls," *The Tenth ACM Symposium on Operating Systems Principles,* 1985.

[15] Clark, D. D. "Host-Network Interface Architecture," *Laboratory for Computer Science, MIT Cambridge*, 1987.

[16] Clos, C. "A Study of Non-blocking Switching Networks" *Bell Syst. Tech. J.*, Vol.32, Mar.1953, pp. 406–424.

[17] Coudreuse, J. P. and M. Servel. "Asynchronous Time-Division Techniques: An Experimental Packet Network Integrating Videocommunication," *Proceedings of the International Switching Symposium*, 1984.

[18] Coudreuse, J. P. and M. Servel. "Prelude: An Asynchronous Time-Division Switched Network," *International Communications Conference*, 1987.

[19] Day, C., J. N. Giacopelli and J. Hickey. "Applications of Self-Routing Switches to LATA Fiber Optic Networks," *International Switching Symposium*, 1987.

[20] De Prycker, M., and J. Bauwens. "A Switching Exchange for an Asynchronous Time Division Based Network," *International Communications Conference*, 1987.

[21] Dias, D. M. and J. R. Jump, "Packet Switching Interconnection Networks For Modular Systems," *Computer*, vol. 14, no. 12, 12/81, 43-53

[22] 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

[23] Dias, D. M. and Manoj Kumar. "Packet Switching in $N \log N$ Multistage Networks," *Proceedings of Globecom 84*, 12/84, 114–120. "Switching Techniques Review for Asynchronous Time

[24] Dieudonne, M. and M. Quinquis. Division Multiplexing," *International Switching Symposium*, 3/87.

[25] Elnahas, S. E., K. Tzou, J. R. Cox, Jr., R. L. Hill and R. G. Jost. "Progressive Coding and Transmission of Digital Diagnostic Pictures," Washington University Computer Science Department, WUCS-86-12, 6/86.

[26] Erdos, P. and A. Renyi. "On random graphs I," *Publication Mathematica Debrecen*, 6:290–297, 1959.

[27] Feldman, P., J. Friedman and N. Pippenger "Non-Blocking Networks" *Proceedings of STOC 1986* 5/86, pp. 247–254.

[28] Feng, Tse-yun and Chuan-lin Wu, "Fault-Diagnosis for a Class of Multistage Interconnection Networks," *EEE Transactions on Computers*, vol. c-30, no. 10, 10/83.

[29] Feng, Tse-yun. "A Survey of Interconnection Networks," *Computer*, vol. 14, no. 12, 12/83, 12–30.

[30] 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.

[31] Gabow, Harold N. *Using Euler Partitions to Edge Color Bipartite Multigraphs International Journal of Computer and Information Sciences*, Vol. 5, No. 4, 1976.

[32] Gilbert, E. N. and H.O. Pollak. "Steiner Minimal Trees." *SIAM J. Appl. Math.*, 16(1):1–29, 1968.

[33] 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.

[34] 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.

[35] 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.

[36] Haserodt, Kurt and Jonathan Turner. "An Architecture for Connection Management in a Broadcast Packet Network," Washington University Computer Science Department, WUCS-87-3.

[37] Hayward, G., L. Linnell, D. Mahoney and L. Smoot. "A Broadband Local Access System Using Emerging Technology Components," *International Switching Symposium*, 3/87.

[38] 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.

[39] Huang, Alan and Scott Knauer. "Starlite: a Wideband Digital Switch," *Proceedings of Globecom 84*, 12/84, 121–125.

[40] Huang, Alan. "Distributed Prioritized Concentrator," U.S. Patent 4,472,801, 1984.

[41] Huang, Alan and Scott Knauer. "Wideband Digital Switching Network," U.S. Patent 4,542,497, 1985.

[42] Imase, Makoto and Bernard Waxman. "Worst-case Performance of Rayward-Smith's Steiner Tree Heuristic," Washington University Computer Science Department, WUCS-88-13.

[43] Jaffe, J. M. "Distributed multi-destination routing: the constraints of local information." *SIAM J. Comput*, 14(4):875–88, 11/85.

[44] Jain, J. R. and A. K. Jain. "Displacement Measurement and Its Application in Interframe Image Coding," *IEEE Transactions on Communications*, vol. COM-29, 1799–1808, 12/81.

[45] 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.

[46] Judice, C. N. and M. J. Jaquez. "High Compression Coding of Entertainment Video for Packet Networks," *IEEE COMSOC International Workshop on Future Prospects of Burst/Packetized Multimedial Communications*, 11/87.

[47] 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.

[48] Kermani, P. and L. Kleinrock. "Virtual Cut-Through: A New Computer Communication Switching Technique," *Computer Networks*, vol. 3, 1979, 267–286.

[49] Khakoo, Shabbir and Jonathan Turner. "System Testing of a Broadcast Packet Switch," Washington University Computer Science Department, WUCS-87-4.

[50] Khakoo, Shabbir. "Improved Search Algorithms for Video Codecs," Washington University Electrical Engineering Department, MS thesis, June 1988.

[51] Kirkpatrick, D. G., M. Klawe and N. Pippenger "Some Graph-Coloring Theorems with Applications to Generalized Connection Networks", *SIAM J. Alg. Disc. Meth.*, Vol. 6, No. 4, Oct.1985, pp. 576–582.

[52] Kou, L., G. Markowsky, and L. Berman. "A fast algorithm for Steiner trees." *Acta Informatica*, 15:141–5, 81.

[53] 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.

[54] Kulzer, John J. and Warren A. Montgomery. "Statistical Switching Architectures for Future Services," *Proceedings of the International Switching Symposium*, 5/84.

[55] Lea, Chin-tau. "The Load-Sharing Banyan Network," *IEEE Transactions on Computers*, 12/86.

[56] Lee, K. Y. "A New Benes Network Control Algorithm", *IEEE Transactions on Computers*, Vol.C-36, No.6, June 1987, pp. 768–772.

[57] Lee, Tony T. "Non-Blocking Copy Networks for Multicast Packet Switching," Bell Communications Research, 1987.

[58] Masson, G. M., Gingher, G. C., Nakamura, S. "A Sampler of Circuit Switching Networks", *Computer*, June 1979, pp.145–161.

[59] 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.

[60] Melen, Riccardo and Jonathan S. Turner. "Nonblocking Multirate Networks," Washington University Computer Science Department, WUCS-88-2, 1/88.

[61] 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.

[62] Montgomery, Warren A. "Techniques for Packet Voice Synchronization," *IEEE Journal on Selected Areas in Communications*, vol. SAC-1, no. 6, 12/83, 1022–1028.

[63] 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.

[64] Nassimi, D. and S. Sahni. "Parallel Permutation and Sorting Algorithms and a new Generalized Connection Network," *JACM*, 1982, 642–667.

[65] Netravali, A. N. and J. D. Robbins. "Motion Compensated Television Coding: Part 1," *Bell System Technical Journal*, vol. 58, No. 3, 3/79.

[66] Netravali, A. N. and J. O. Limb. "Picture Coding: A Review," *Proceedings of the IEEE*, vol. 68, No. 3, 3/80.

[67] Opferman, D. C. and N. T. Tsao-Wu "On a Class of Rearrangeable Switching Networks, Part I: Control Algorithm", *Bell Syst. Tech. J.*, Vol.50, 1971, pp. 1579–1600.

[68] Parulkar, Guru and Jonathan Turner. "Towards a Framework for High Speed Communication in a Heterogeneous Networking Environment," Washington University Computer Science Department, WUCS-88-7.

[69] Patel, J. H. "Performance of Processor-Memory Interconnections for Multiprocessors," *IEEE Transactions on Computers*, vol. C-30, no. 10, 10/81, 771-780.

[70] Patel, J.K. "Performance of Processor-Memory Interconnections for Multiprocessors", *IEEE Transactions on Computers*, Vol.C-30, No.10, Oct.1981, pp. 301–310.

[71] Pippenger, N. "The Complexity of Switching Networks," MIT, PhD thesis, 1973.

[72] Pippenger, N. "Telephone Switching Networks", *Proceedings of Symposia in Applied Mathematics*, Vol. 26, 1982, pp.101–133.

[73] Postel, J. "DOD Standard Internet Protocol," *ARPA RFC 760*

[74] Postel, J. "DOD Standard Transmission Control Protocol," *ARPA RFC 761*

[75] Pratt, W. K. "Image Transmission Techniques," Wiley (Interscience), New York, 1978.

[76] 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.

[77] 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.

[78] Richards, Gaylord and Frank K. Hwang. "A Two Stage Rearrangeable Broadcast Switching Network," *IEEE Transactions on Communications*, 10/85.

[79] Robbert, George. "Design of a Broadcast Translation Chip," Washington University Computer Science Department, WUCS-87-9.

[80] Robbert, George. "A Circuit Generator for Synchronous Streams Processors," Washington University Computer Science Department, MS thesis, May 1988.

[81] Sincoskie, W. D. "Transparent Interconnection of Broadcast Networks," *Proceedings of the International Zurich Seminar on Digital Communication*, 3/86, 131–134.

[82] Shannon, C. E. "Memory Requirements in a Telephone Exchange", *Bell Syst. Tech. J.*, Vol.29, 1950, pp.343–349.

[83] Srinivasan, R. and K. R. Rao. "Predictive Coding Based on Efficient Motion Estimation," *Proceedings of the International Communications Conference*, 521–526, 5/84.

[84] 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.

[85] Sterbenz, James. "Design of a VLSI Packet Switch Element," Washington University Computer Science Department, WUCS-88-5.

[86] Stroustrup, Bjarne, "The C++ Programming Language," Addison-Wesley, 1986.

[87] Thompson, C. "Generalized Connection Networks for Parallel Processor Intercommunication," *IEEE Transactions on Computers*, 12/78, 1119–1125.

[88] 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.

[89] Turner, Jonathan S. and Leonard F. Wyatt. "A Packet Network Architecture for Integrated Services," *Proceedings of Globecom 83*, 11/83, 45–50.

[90] Turner, Jonathan S. "Fast Packet Switching System," United States Patent #4,494,230, 1/15/85.

[91] Turner, Jonathan S. "Design of an Integrated Services *Packet* Network," *Proceedings of the Ninth ACM Data Communications Symposium*, 9/85, pages 124–133.

[92] 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.

[93] Turner, Jonathan S. "Design of a Broadcast Packet Network," *Proceedings of Infocom*, 4/86.

[94] Turner, Jonathan S. "New Directions in Communications," *IEEE Communications Magazine*, 10/86.

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

[96] Turner, Jonathan S. "Specification of Integrated Circuits for a Broadcast Packet Network," Washington University Computer Science Department, WUCS-87-5.

[97] 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.

[98] Turner, Jonathan S, "Fluid Flow Loading Analysis of Packet Switching Networks," Washington University Computer Science Department, WUCS-87-16, 7/87.

[99] Turner, Jonathan, "Buffer Management System," Washington University Computer Science Department, WUCS-88-6.

[100] Turner, Jonathan, "Broadcast Packet Switching Network," Unites States Patent #4,734,907, March 1988.

[101] Verbeist, W. "Video Coding in an ATD Environment," *Picture Coding Symposium*, 6/87.

[102] Waxman, Bernard. "Thesis Proposal: Routing of Multipoint Connections," Washington University Computer Science Department, WUCS-87-2.

[103] Waxman, Bernard. "Probable Performance of Steiner Tree Algorithms," Washington University Computer Science Department, WUCS-88-4.

[104] Wirsching, J. E. and T. Kishi, "Conet: A Connection Network Model," *IEEE Transactions on Computers*, vol. C-30, 4/81.

[105] 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.