Report Number: WUCS-88-28

1988-08-31

Advanced Communications Systems

Jonathan S. Turner

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 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 network 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 Service... Read complete abstract on page 2.

Advanced Communications Systems

Jonathan S. Turner

Complete Abstract:

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 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 network 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 Service 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 applications-dependent functions so as to provide maximum flexibility for future services.

ADVANCED COMMUNICATIONS SYSTEMS

Jonathan S. Turner

WUCS-88-28

September 1, 1987 - August 31, 1988

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

with 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 and 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

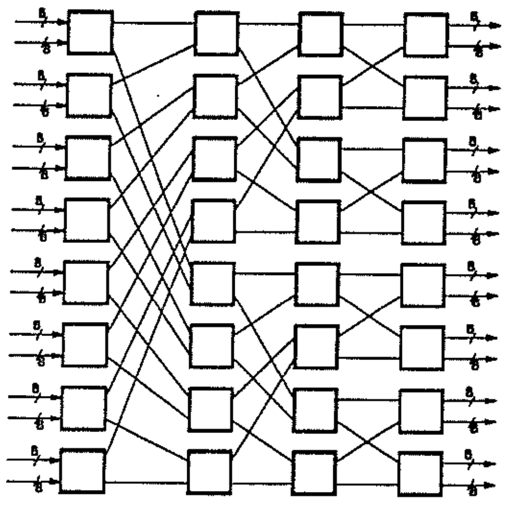
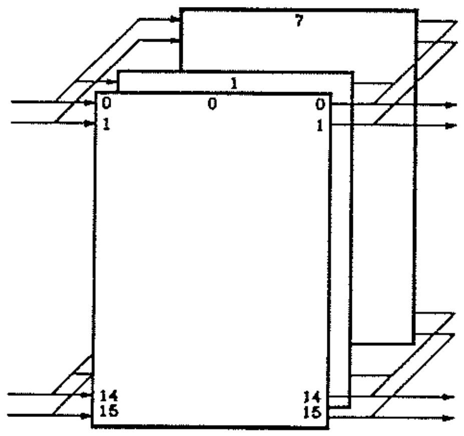

Figure 2.1: Comparison of Word-Serial and Bit-Sliced Organizations

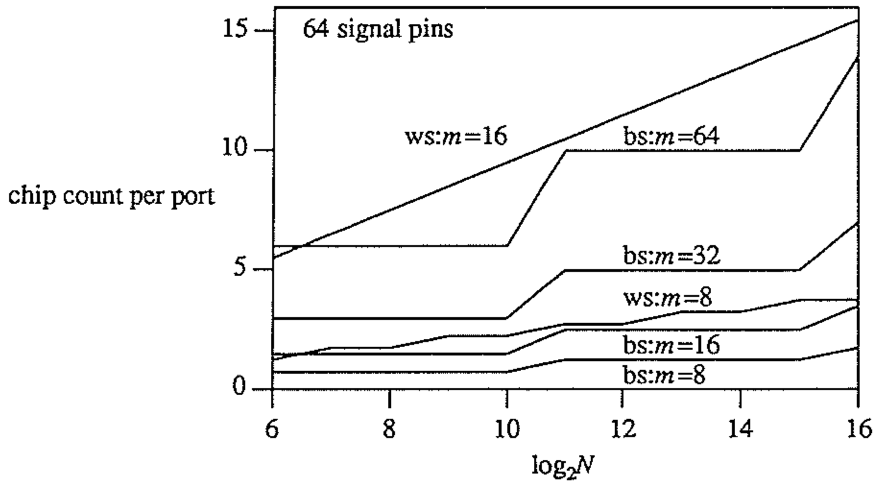

Figure 2.2: Complexity of Word-Serial and Bit-Sliced Beneš Networks

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

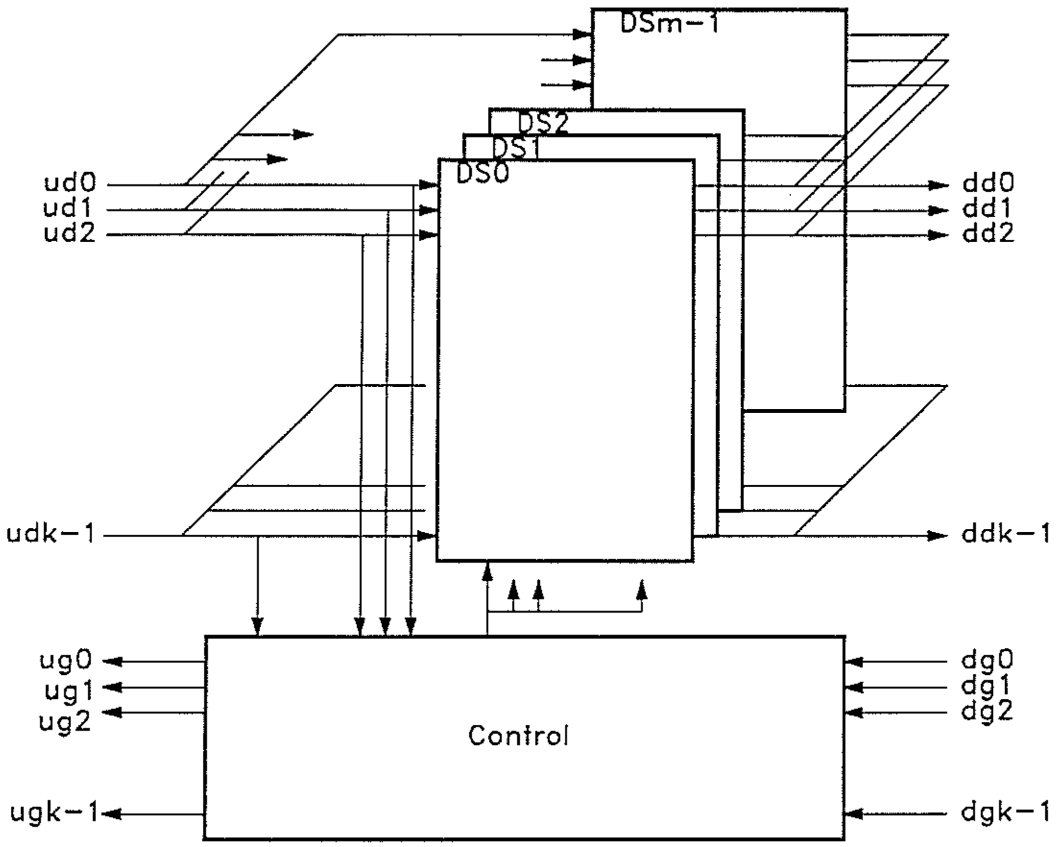

Figure 2.3: Bit-Sliced Switch Element

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.

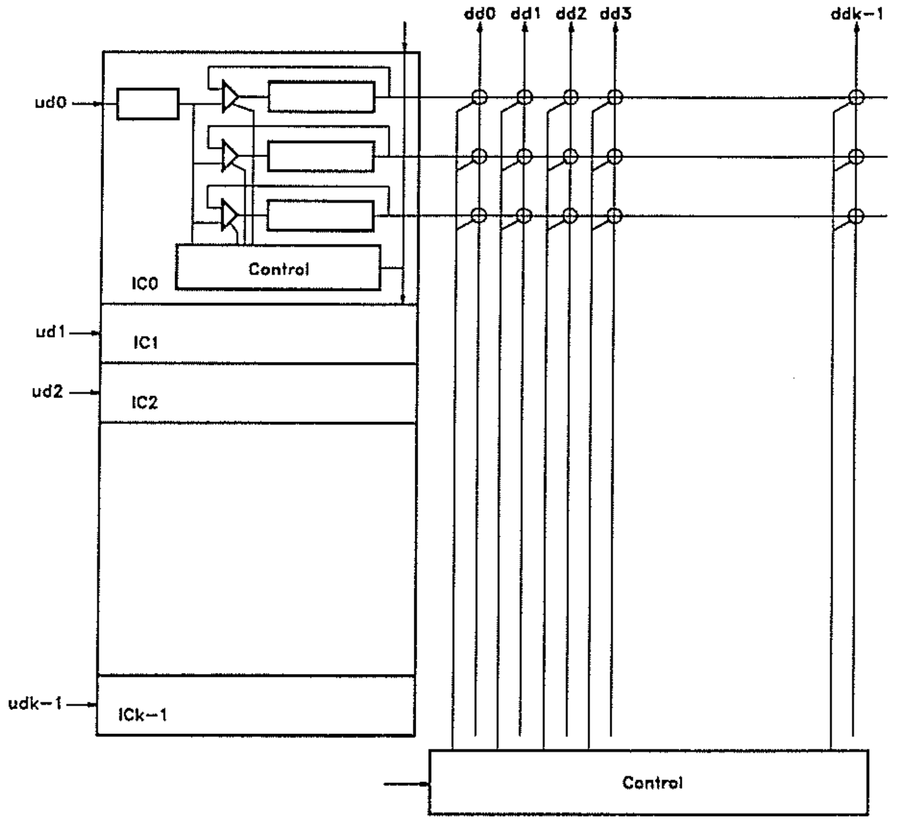

Figure 2.4: Data Slice with Input Buffering

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

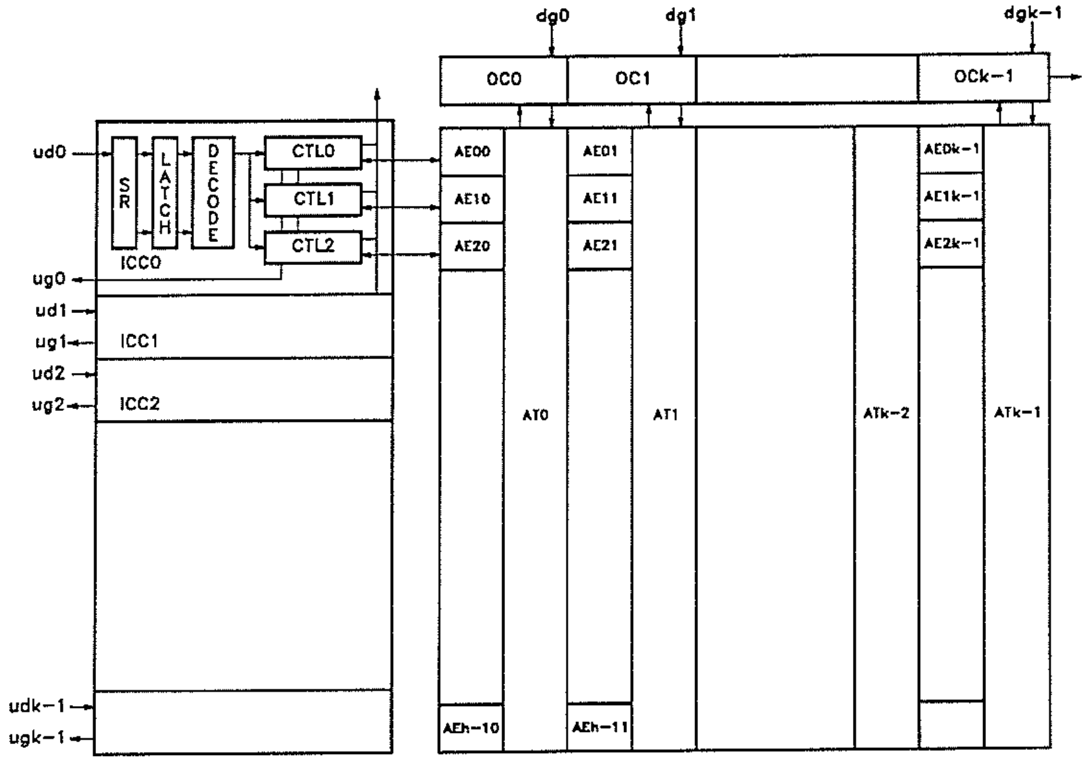

Figure 2.5: Control Slice

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,

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

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

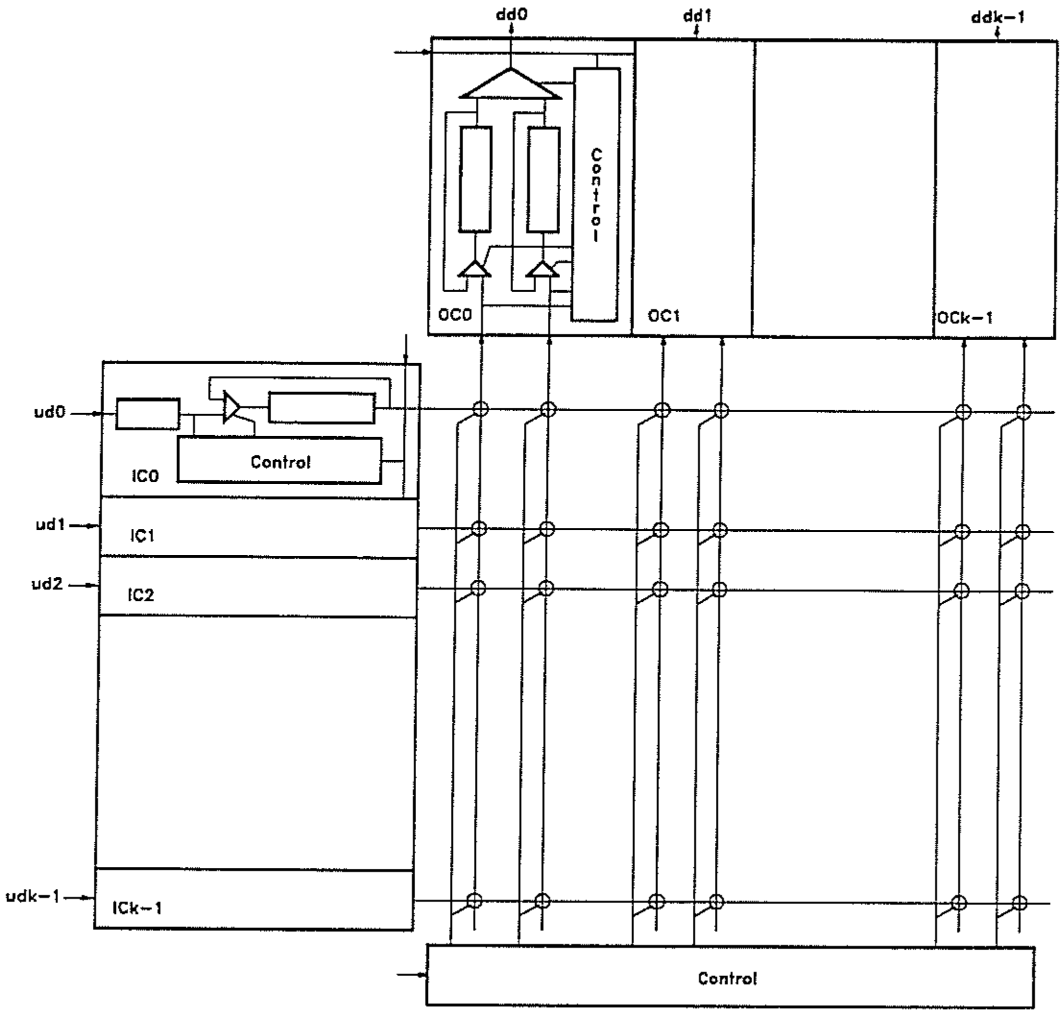

Figure 2.6: Data Slice with Output Buffering

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-

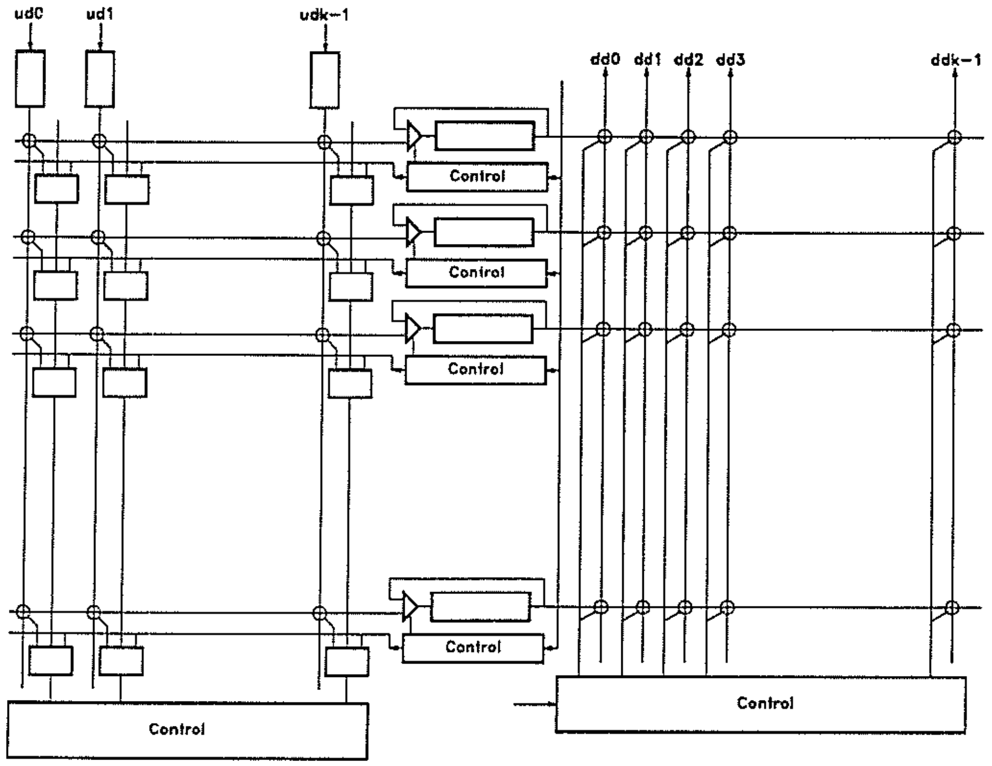

Figure 2.7: Data Slice with Shared Buffering

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

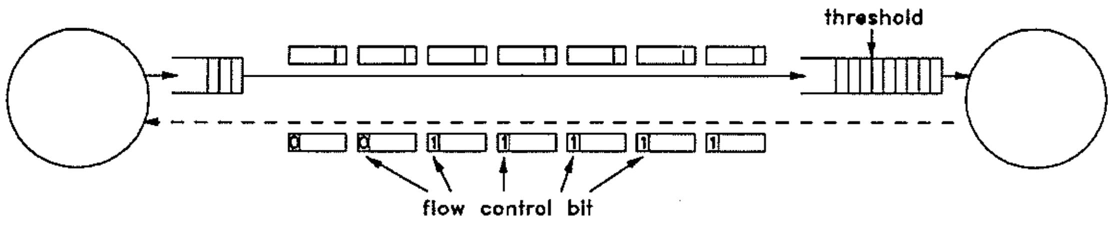

Figure 2.8: Simple Flow Control Across a Link

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

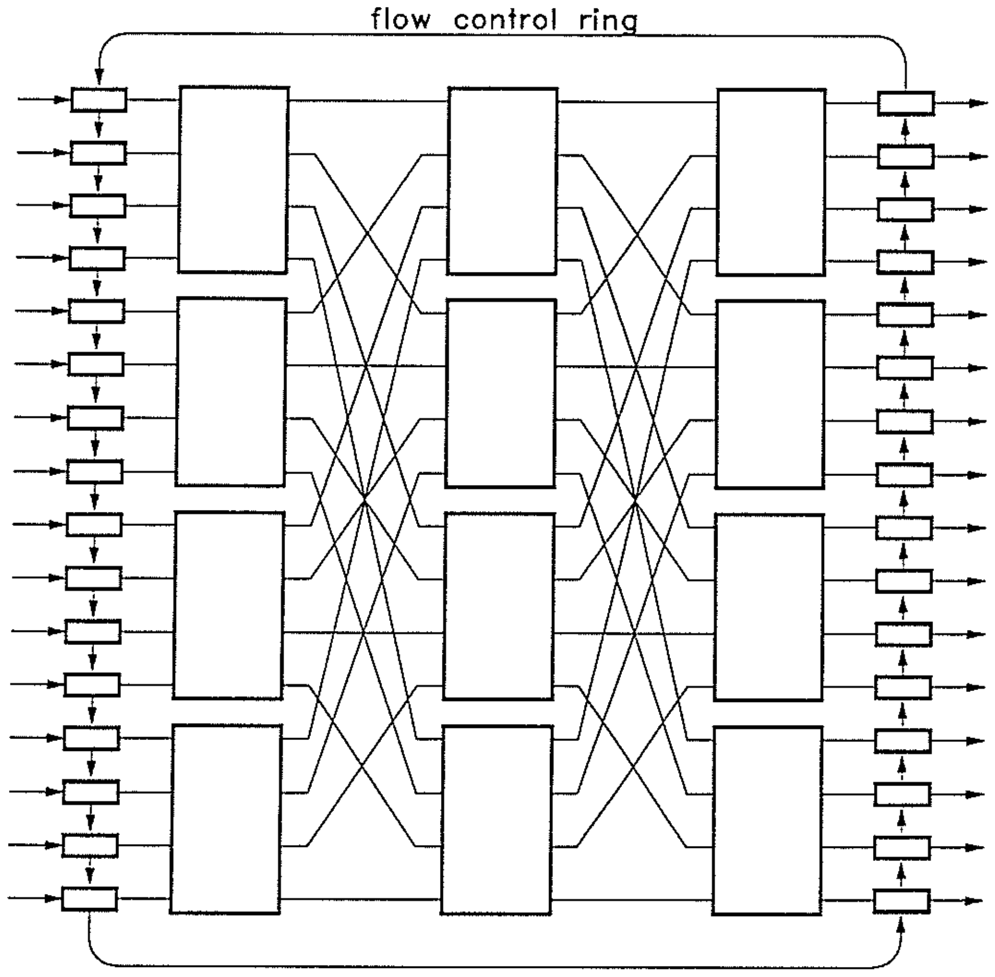

Figure 2.9: Flow Control Across a Switch

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

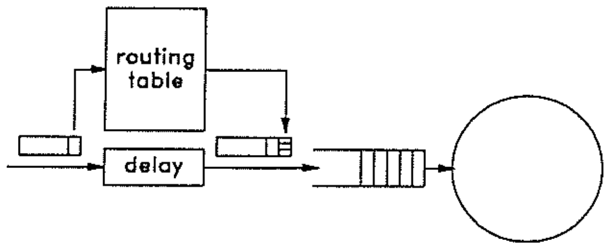

Figure 2.10: Datagram Address Translation

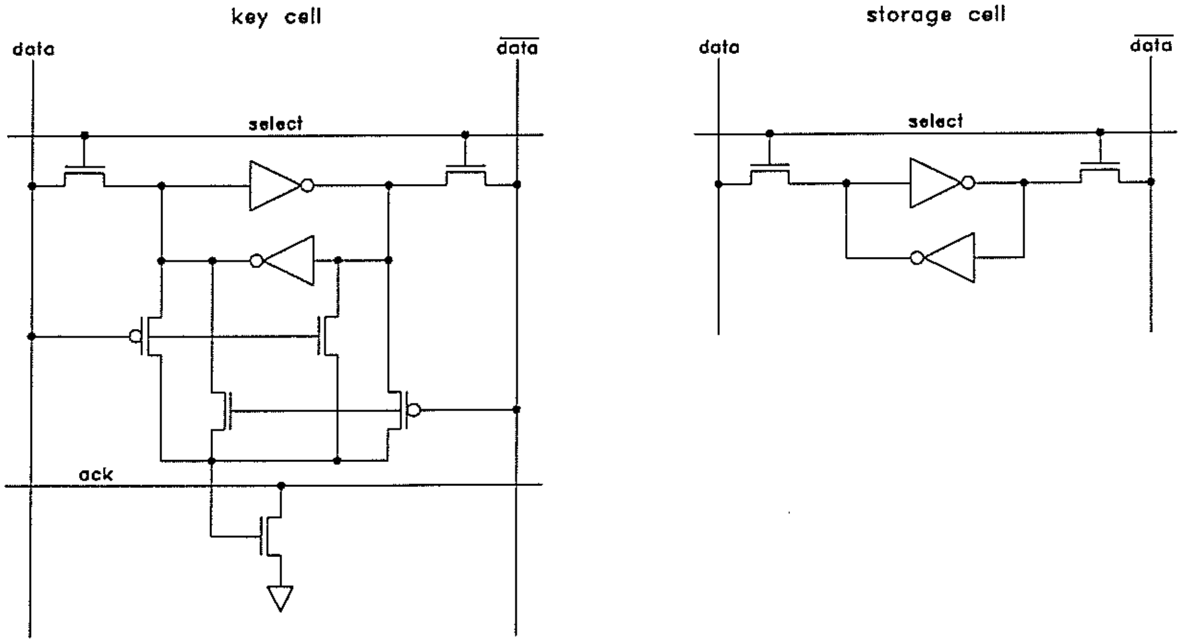

Figure 2.11: Content Addressable Memory Cells

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

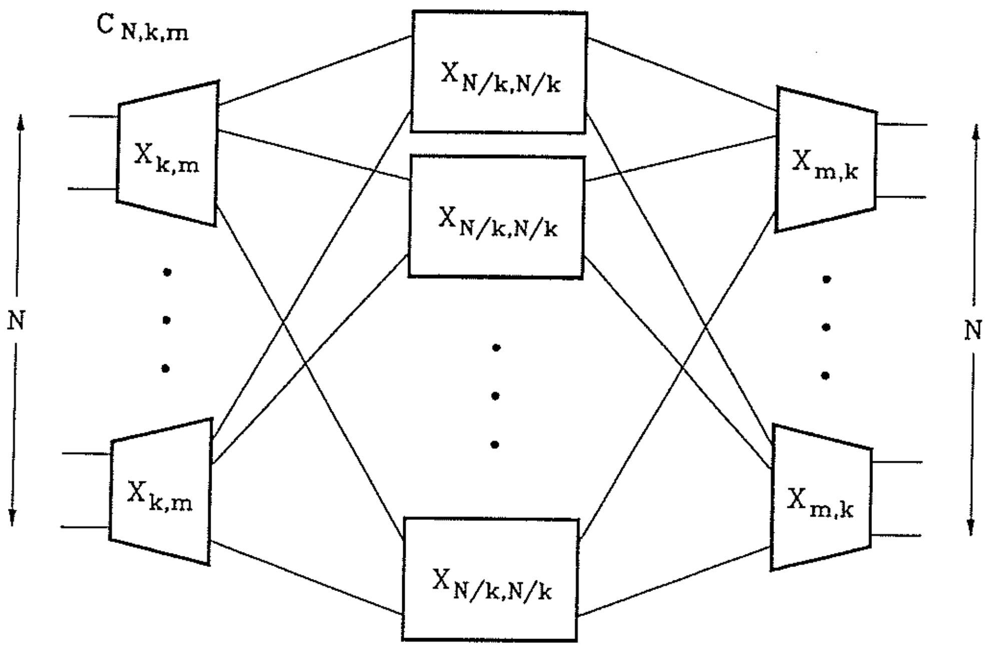

Figure 3.1: Clos Network

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*

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

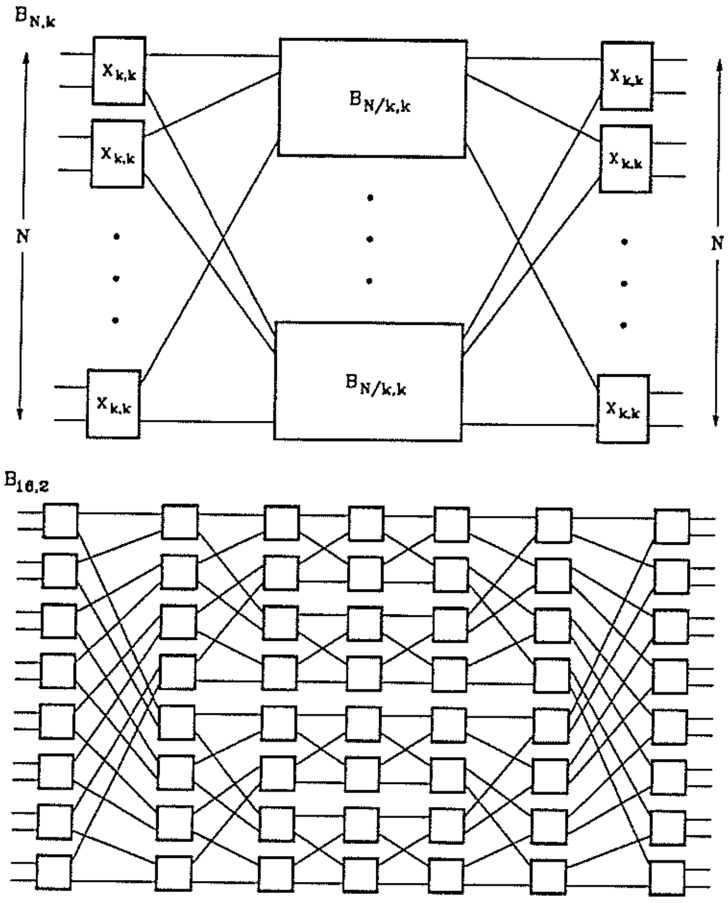

Figure 3.2: Beneš Network ($B_{N,k}$)

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 $\leq 1/2$

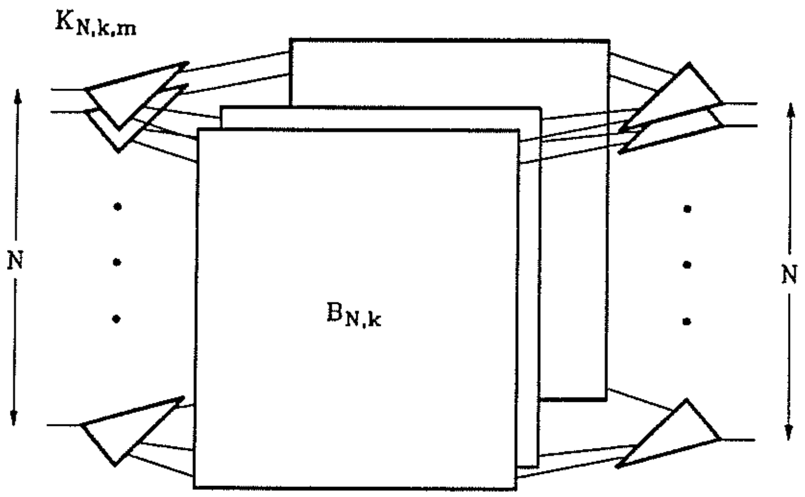

Figure 3.3: Cantor Network

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*

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

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

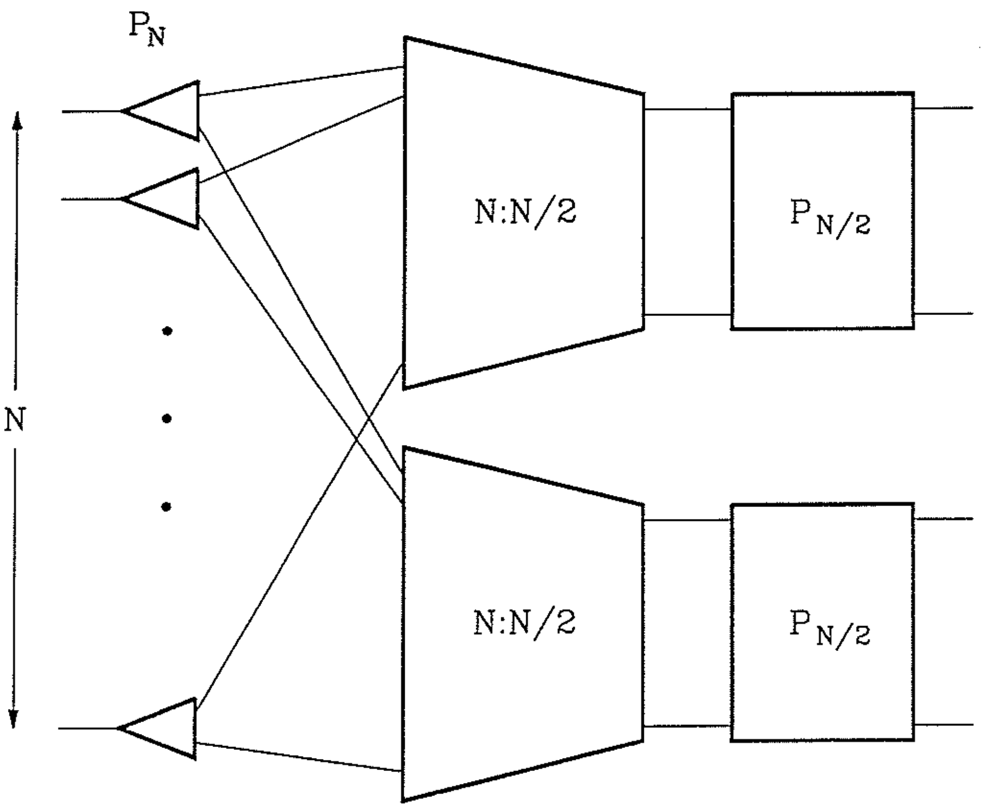

Figure 3.4: Recursive Construction of P_N

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.

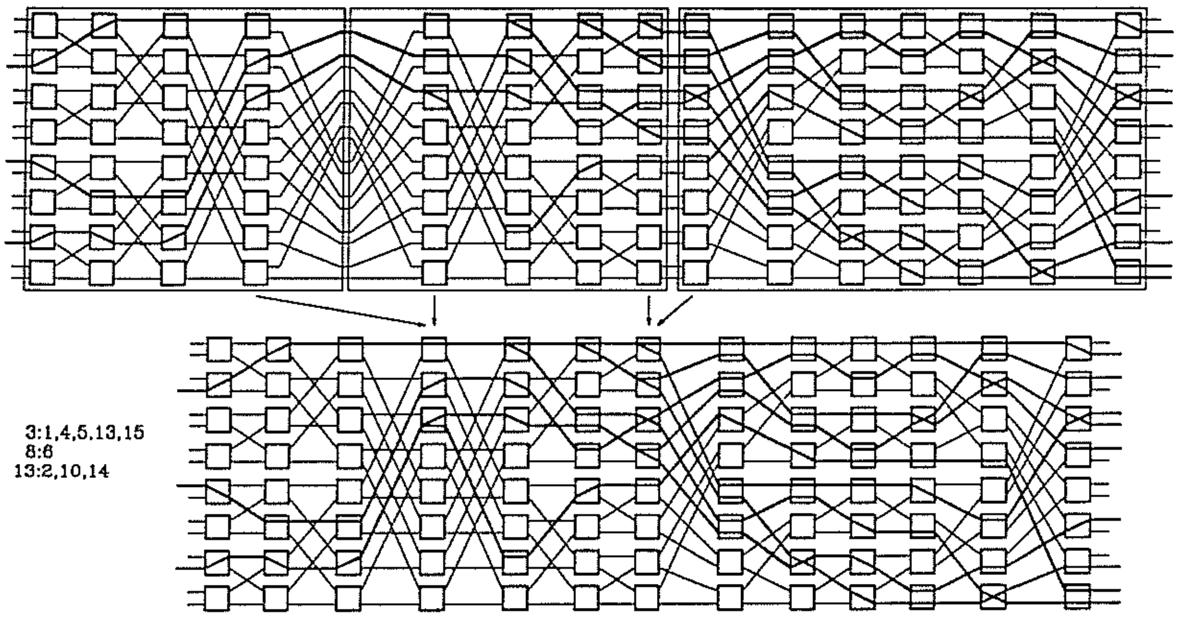

Figure 3.5: Example of Routing in $T_{N,2}$

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*

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

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

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

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

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

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

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

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* $\mathcal{C}$ 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

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

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

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

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

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

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

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

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.

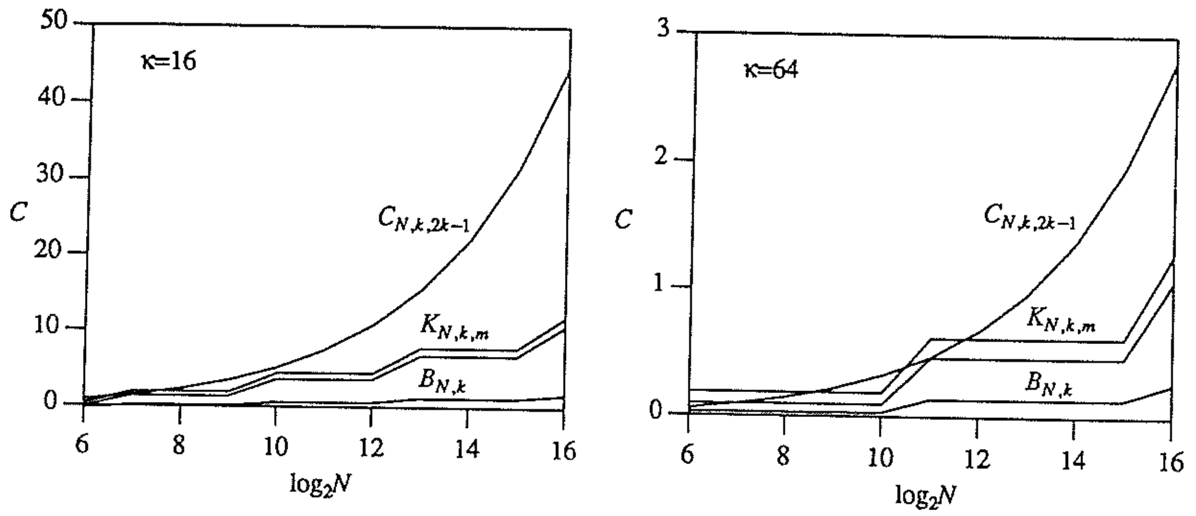

Figure 3.6: Complexity of Multirate Networks: Beneš is shown for rearrangeable and strictly nonblocking case. $\beta = 1/2$ for Clos and Kantor.

For the Cantor network,

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

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

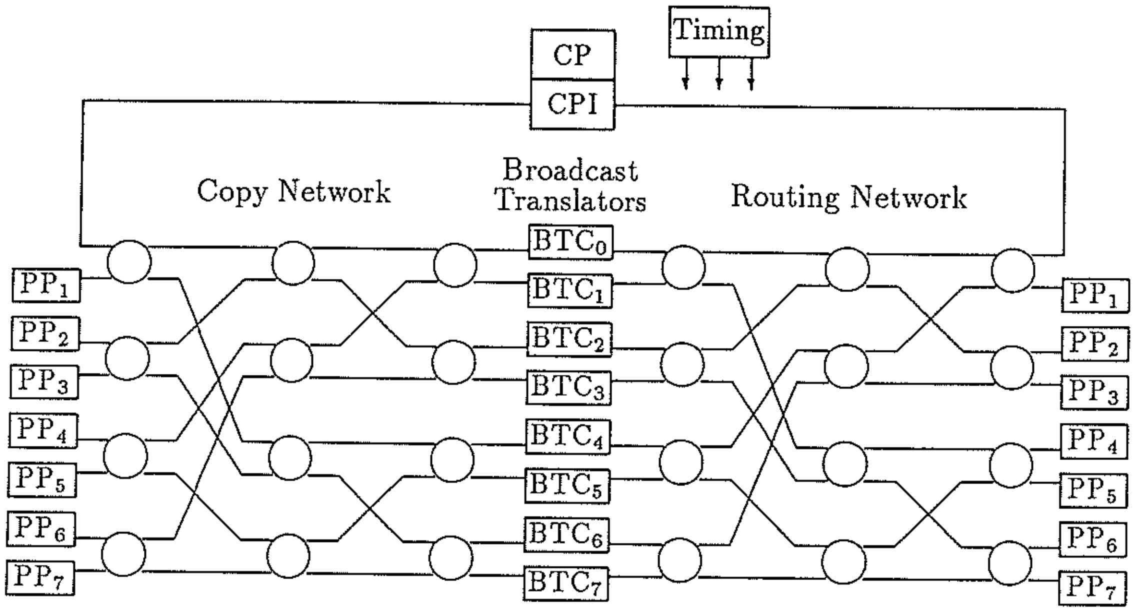

Figure 4.1: Prototype Switch Module

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.

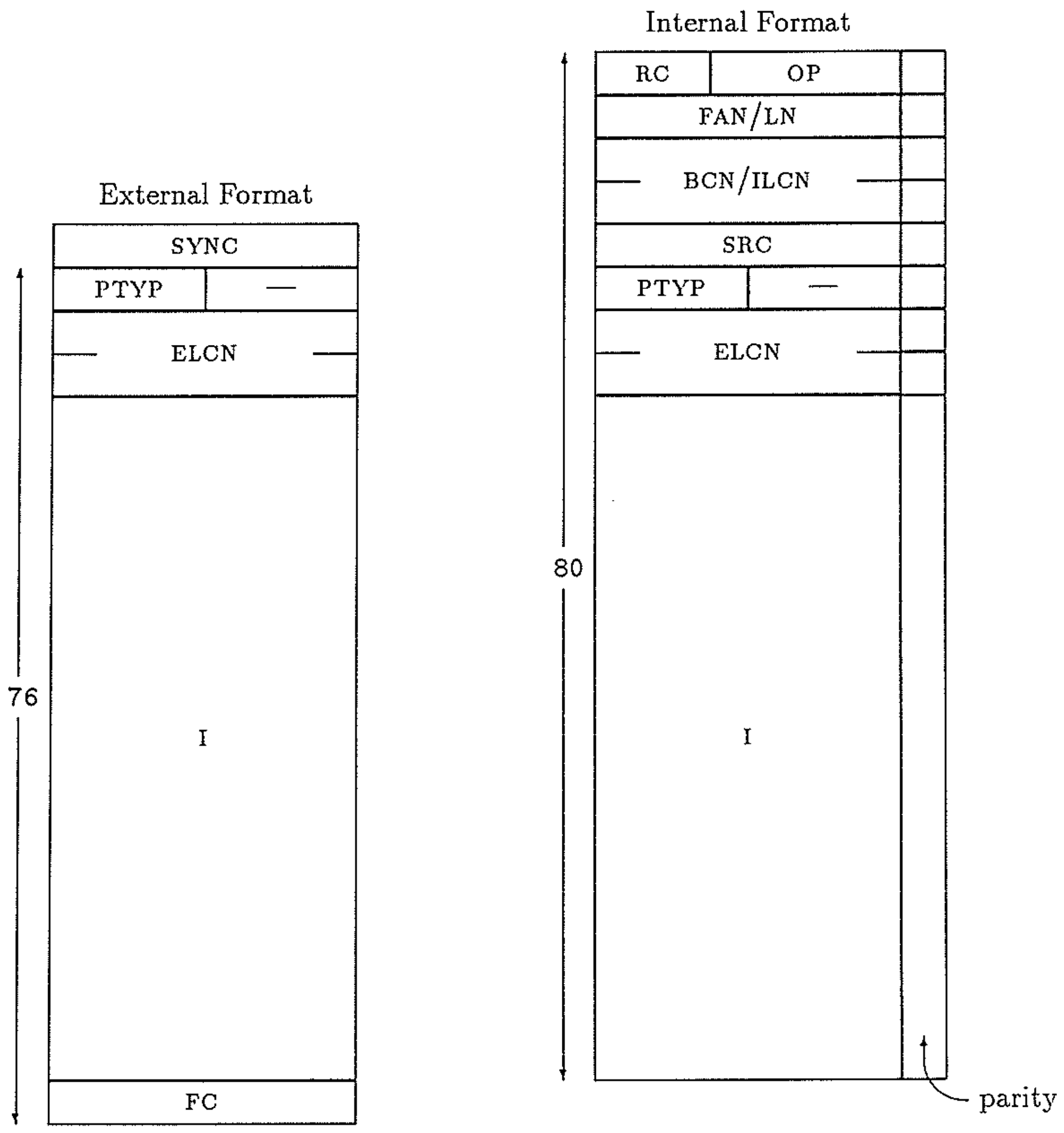

Figure 4.2: Packet Formats

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

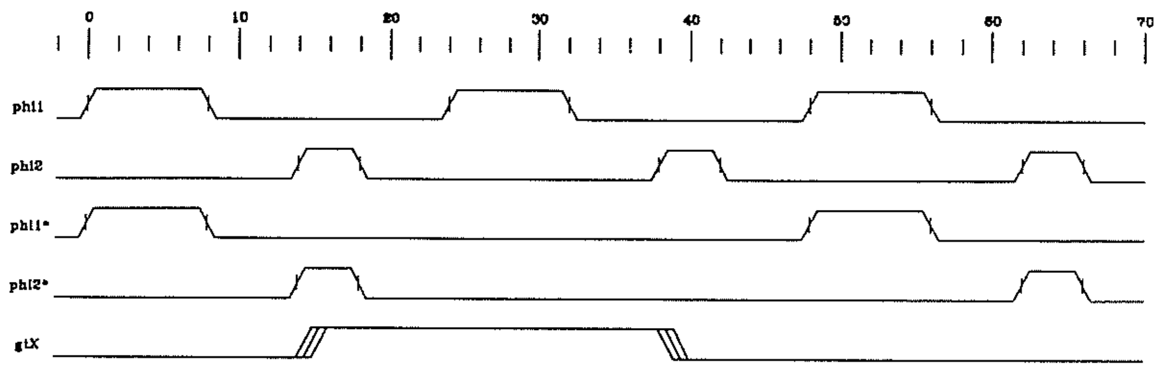

Figure 4.4: Global Clock Signals

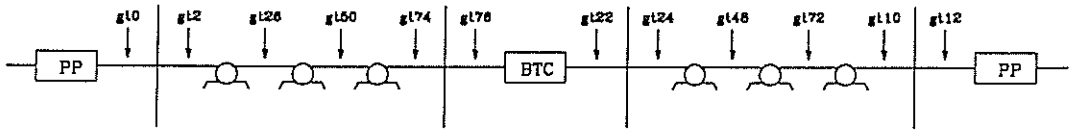

Figure 4.5: Global Timing Relationships

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."

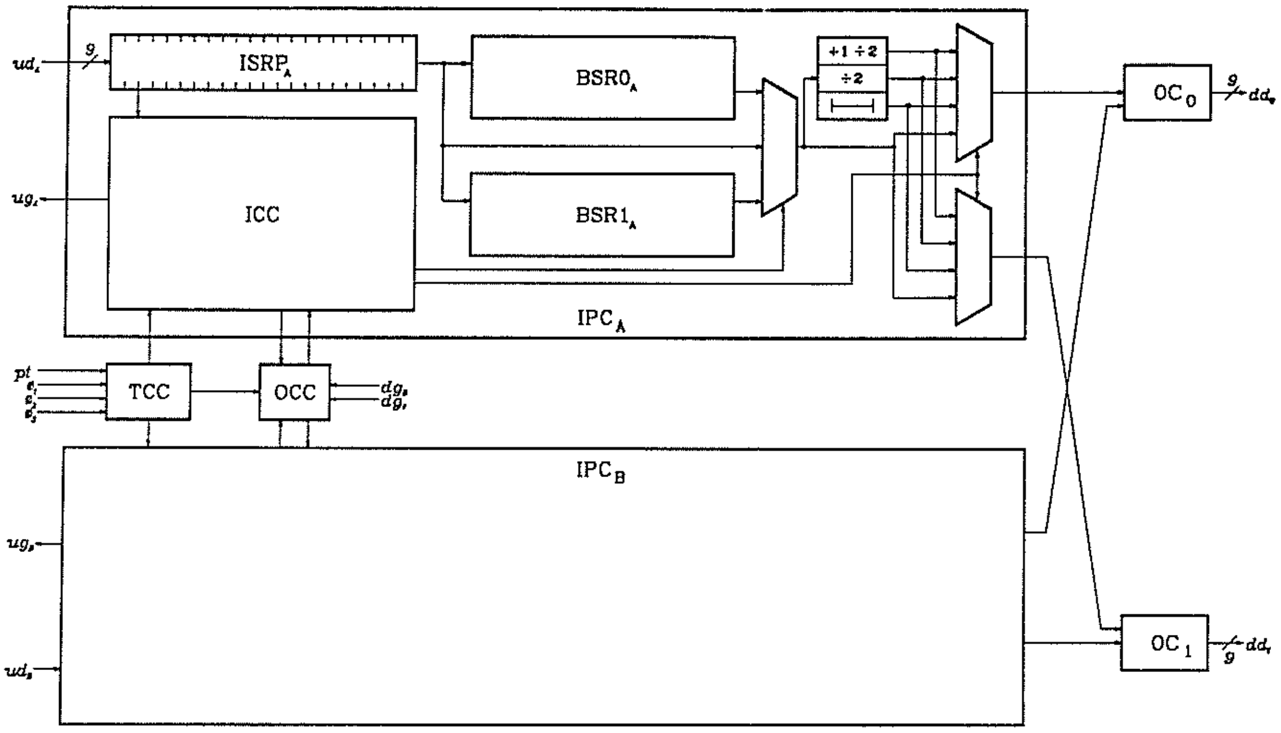

Figure 4.6: Block Diagram of Packet Switch Element Chip

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 `t`i. 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 `t`i and `t`i`:`j, for various values of i, j. Signal `t`i is high during clock period `t`i of the epoch; in particular it goes high during `phi2` of the preceding clock cycle and goes low before `phi2` goes high again. Signal `t`i`:`j is similar; it is high during `t`i and stays high through `t`j.

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

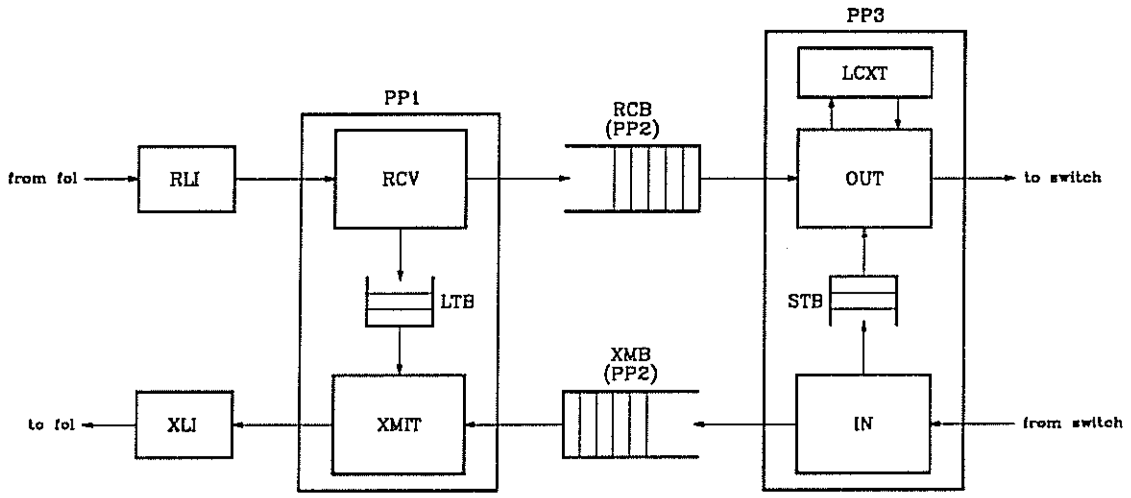

Figure 4.7: Packet Processor Block Diagram

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.

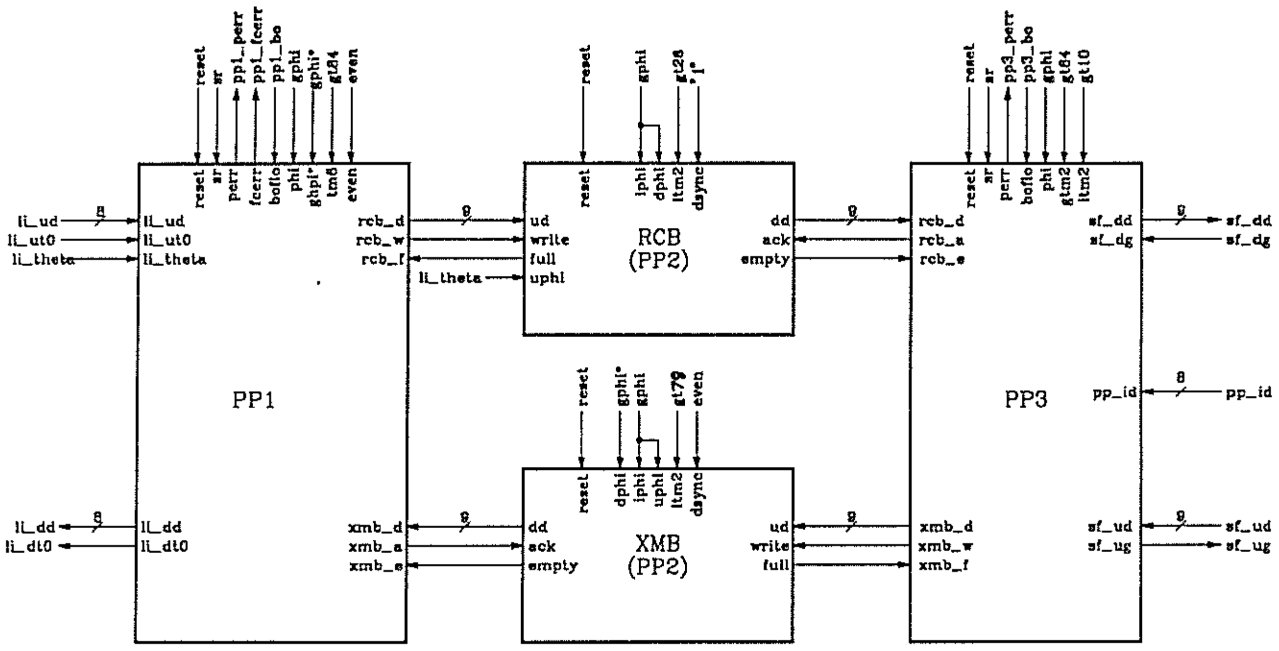

Figure 4.8: Packet Processor Signals

- *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

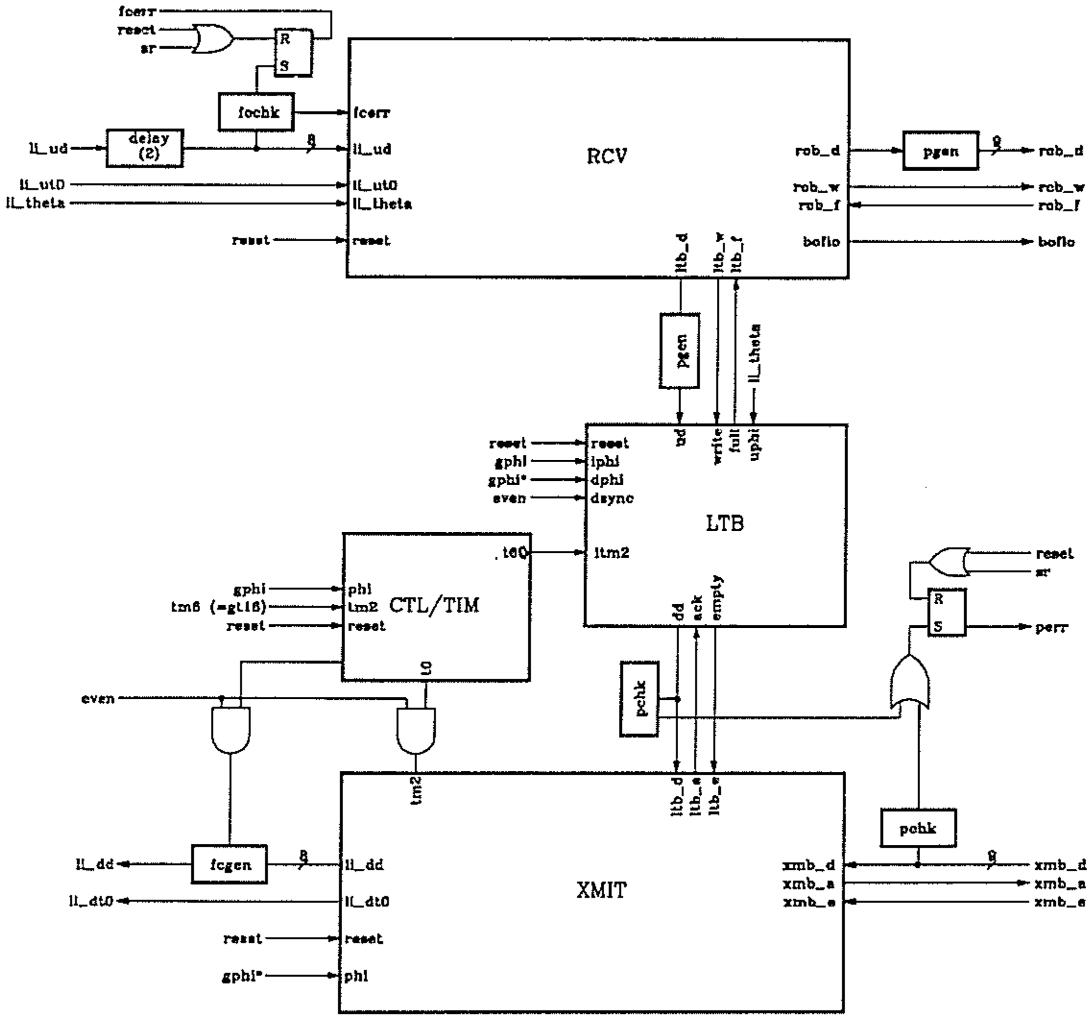

Figure 4.9: PP1 Block Diagram

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.

```
rcv@86( port[8] ext_pkt <li_ud@0;        /* data from link interface */

        port[8] ext_pkt >rcb_d@4;        /* data to rcb              */
        port[1] bit     >rcb_w@80;       /* rcb write signal         */
        port[1] bit     <rcb_f@0;        /* rcb full signal          */

        port[8] ext_pkt >ltb_d@4;        /* data to ltb              */
        port[1] bit     >ltb_w@80;       /* ltb write signal         */
        port[1] bit     <ltb_f@0;        /* ltb full signal          */

        port[1] bit     <fcerr@75;       /* frame check error        */
        port[1] bit     >boflo@80)       /* 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