THE TOPOLOGY, ALGORITHMS AND ANALYSIS OF A SYNCHRONOUS OPTICAL HYPERGRAPH ARCHITECTURE

BY

YORAM OFEK

B.S., Technion, Israel Institute of Technology, 1979
M.S., University of Illinois, 1985

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1987

Urbana, Illinois

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

THE GRADUATE COLLEGE

MAY 1987

WE HEREBY RECOMMEND THAT THE THESIS BY

YORAM OFEK

ENTITLED THE TOPOLOGY, ALGORITHMS AND ANALYSIS OF

A SYNCHRONOUS OPTICAL HYPERGRAPH ARCHITECTURE

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF DOCTOR OF PHILOSOPHY

[illegible]
Director of Thesis Research

[illegible]
Head of Department

Committee on Final Examination†

[illegible]
Chairperson

[illegible]

[illegible]

[illegible]

[illegible]

† Required for doctor's degree but not for master's.

O-517

THE TOPOLOGY, ALGORITHMS AND ANALYSIS OF A SYNCHRONOUS OPTICAL HYPERGRAPH ARCHITECTURE

Yoram Ofek, Ph.D.
Department of Electrical and Computer Engineering
University of Illinois at Urbana–Champaign, 1987

One of the main challenges in computer engineering today is the question of how to connect a large number of computing nodes, which are distributed over a large area, into one integrated system. The functions performed by the system are varied from electronic mail, distributed file servers to parallel processing, and real–time data communication (e.g., voice and video).

The research, presented in thesis, involves a bottom–up design of a fiber optic hypergraph network. The network is constructed of nets, which are passive optical stars, and constitute a multiple access medium. The topology and the high bandwidth (>1 gigabit/second) enable the construction of a low–dimension, synchronous, hypergraph network. The system maintains a global event synchronization with total ordering and uniformly integrates various functions by using distributed algorithms. It will be shown how voice and data are integrated on the system and how time stamps are used for implementing concurrency control algorithms (e.g., mutual exclusion). These algorithms are designed for a real–time operation in a very high bandwidth.

The following new ideas were incorporated:

• a new family of *conservative codes*, for preserving time integrity and decoding, without a phase–locked loop

• a hierarchical interface to one gigabit/second serial link

• a protocol for periodic exchange of state and time information, which enables the implementation of a hybrid multiple–access control scheme, and the integration of other

network control functions

- an algorithm for maintaining global event synchronization and total event ordering
- a scheme for integrating voice and data by constructing virtual multiplexers and demultiplexers
- an *isolate and skip* mechanism for graceful degradation of the network performance
- a partial hypergraph as a centralized switch

DEDICATION

To Barbara, and Our Children

Tidhar, Gidon, and Daphna

ACKNOWLEDGEMENTS

This work was influenced by many people, their thoughts, and their work, which in turn stimulated my own ideas. I would like to thank my advisor Professor Michael Faiman for his fruitful advice and complete support. I thank my thesis committee: Professors J. Liu, E. Davidson, W. Kubitz and P. Banerjee. I also thank Phil McKinley and Rajagopalan for interesting discussions on the optical hypergraph.

I thank the librarians in the Engineering Library for their assistance and advice during my graduate studies. The computer services in the Department of Computer Science facilitated a very effective environment for research; I would like to thank everyone involved in this operation.

I would like to thank my wife Barbara for her invaluable assistance. Finally, I thank my parents, Yehudith and Yaacov, from whom I acquired the art of critical reading and the love for books.

TABLE OF CONTENTS

"If you know a mathematical proposition, that's not to say you yet know anything. If there is confusion in our operations, if everyone calculates differently, and each one differently at different times, then there isn't any calculating yet; if we agree, then we have only set our watches, but not yet measured any time.

If you know a mathematical proposition, that's not to say you yet know anything.

I.e., the mathematical proposition is only supposed to supply a framework for a description."

(Ludwig Wittgenstein, "**Remarks on the Foundation of Mathematics**").

CHAPTER 1.

INTRODUCTION

1.1. Overview

Presented in this thesis a design (description), reasoning, and analysis of a large area network, which is based on the hypergraph topology. One of the main challenges in computer engineering today is the question of how to connect a large number of computing nodes, which are distributed over a large area, into one integrated system. The functions performed by the system may vary from electronic mail and distributed file servers to parallel processing and real–time data communication (e.g., voice and video). The proposed hypergraph network is a realistic system architecture for a very large network, and can perform the above functions efficiently and in a distributed manner.

The work in this research is part of an AT&T project for developing a metropolitan area network. This term denotes a network having thousands of nodes within an area of about 1000 square kilometers and integrates the transmission of data, voice, and video. It will be based on high bandwidth, single mode, and optical fiber links.

The network topology is a hypergraph, which is a set of nets; each net is a set of two or more nodes [Ullm84]. The nets form the edges of a hypergraph, and the nodes are its vertices. The vertices of a particular edge have complete connectivity, so that a message can be transferred between any pair of its vertices. The nets or edges can be viewed also as buses, and each node has ports to several buses (at least one). This type of system is often called a bus–based topology or bus–connected architecture [RePa85].

One of the motivations for this research is to exploit recent technological advances, which make it possible to construct a high–performance, large–area network. The design will make use of the advances in the following areas:

(i) Single mode optical fibers, with a very high bandwidth, low attenuation (less than 0.5 db/km), and low spectral–group delay difference [Kapr85]; and low–loss, passive, optical star couplers.

(ii) High–speed logic devices from GaAs and ECL technologies, which make it possible to construct a digital electronic interface to a serial link with a transmission rate of more than one gigabit/second.

(iii) Very high–density custom and semicustom integrated circuits, for executing complex network control algorithms in real time.

An optical fiber is a dielectric waveguide which can transfer electromagnetic energy over long distances with very low losses. Electrical transmission lines have relatively low efficiencies at frequencies higher than a few gigahertz. Optical waveguides are feasible well above these frequencies.

Optical fibers (see [Kapr85] and [Pers83]) have the following properties:

(1) Electrical Isolation – Optical communication enables data transfers among subsystems that are electrically isolated from one another, eliminating the problems of ground loops and ground noise.

(2) Noise Immunity – optical links are immune to most electric and electromagnetic noise sources; e.g., RFI, EMI.

(3) Low Loss – less than 0.5dB/km for single–mode fibers.

(4) Security – A fiber–optic link is difficult to tap without detection, and it is practically impossible to sense the transmitted information (due to its low losses).

(5) Passive optical coupling – This enables electromagnetic energy to be split between two fibers. The star topology is based on optical couplers. It has very good fault tolerant properties, and can continue to operate in the event of multiple failures.

The major motivation for high–speed fiber optic links (over 1 gigabit/second) is to decrease the dimension of the hypergraph, and to increase the width (number of ports from different nodes) on every hypergraph edge. As a result, the overall system's communication and computation control is simpler. It will be shown that it is possible to design high efficiency algorithms for a two– and three–dimensional hypergraph. These algorithms are executed in real time, independent of the distributed operating system software.

Each net of the network is realized as a centralized, passive, optical star. The messages from the net's ports are merged into one small area in space (a few inches), and are then broadcast back to all the net's ports. The centralized net's topology simplifies the synchronization of all the nodes on the net. Time is slotted, such that one time slot is about one message or packet interval. The centralized star topology can tolerate multiple failures and can detect collisions of packets of any size. The simple net synchronization together with the low hypergraph dimension make it possible to achieve global event synchronization, which is very important for implementing distributed functions and integrating real–time data communication.

1.2. Thesis

A low–dimension, a high width (the number of ports on a net), globally synchronized optical hypergraph, can be efficiently (with low overhead) managed and controlled in a distributed manner, such that:

(1) The proposed distributed algorithms (e.g., access control, synchronization, voice integration, overflow prevention) improve their performance as the state/time information propagates faster through the system. Increasingly higher bandwidth makes the slot duration shorter, which makes the state update more often. The high bandwidth enables the construction of a low–dimension hypergraph; therefore, the state information should propagate via, at most, one, two or three nets. The net and network global synchronization will enable a well–defined state transition at the end of each time slot, which simplifies the implementation of distributed algorithms.

(2) Algorithms for access control, global synchronization, routing, buffer management, and voice integration are independent of the distributed operating system. Furthermore, these algorithms are based on one another. Their construction is done bottom–up; a higher–level algorithm is based on lower–level algorithms. Thus, the routing algorithm is based on the properties of the access control and synchronization algorithms, the access control is derived from properties of the encoding/decoding scheme, and the encoding/decoding is based on the properties of the fiber optic communication medium.

(3) The net interface can tolerate permanent and intermittent physical failures by dynamic reconfiguration and adaptation of the network control algorithms. It will be shown that the centralized, passive optical star is very reliable and inherently redundant. The objectives of the fault tolerant mechanism are to minimize the probability of packet loss and to avoid the transmission of packets which cannot reach their destinations. From the reconfigurability of the system, it is possible to derive many applicable variants of the basic system, e.g., from the regular hypergraph it is possible to derive the partial hypergraph, such that both have the same distributed operating principles.

1.3. The Basic Construction Steps

The basic methodology for the design proceeds bottom–up, as shown in Figure 1.1. Using this design methodology it is possible to exploit the unique properties of the optical medium and to guarantee the physical realizability (or feasibility) of the design.

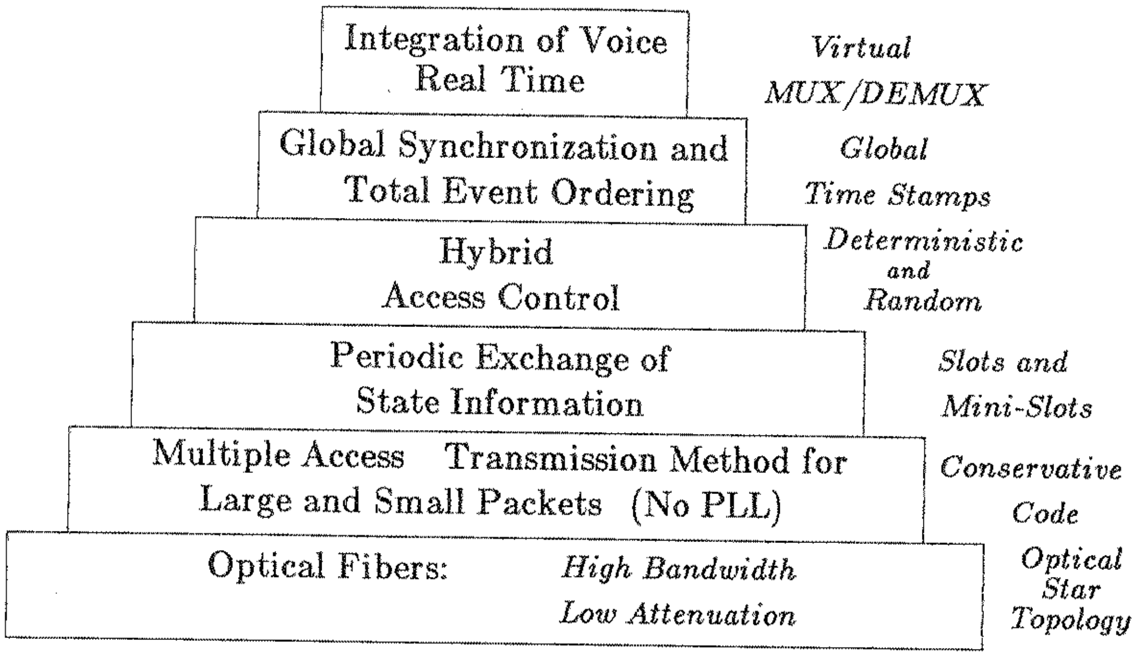


Figure 1.1: Design Methodology

Described in Chapter 2 the basic principles of the system's operation and provides an overview of its structure. The connectivity among the network nodes is defined, and the basic mechanism in which they interact is discussed. In Chapter 3 a new type of encoding scheme is proposed and analyzed. The code is designed to preserve the time information for self-clocking in serial communication over multiple-access channels. This new code is called conservative, since each codeword has a fixed number of transitions and a known delimiting transition at the end. It can be decoded without explicitly recovering the receiving clock with a phase-locked loop (PLL). Thus, using this new coding scheme, it is possible to broadcast short control messages, one after the other, from different and asynchronous nodes. These short control packets are essential for uniformly integrating various system functions. The code is analyzed under various constraints and is shown to be efficient. In Chapter 4 the design of a digital interface is presented, making use of the conservative code. The critical timing path of the interface is optimized for maximum digital bandwidth.

In Chapter 5 a detailed design and analysis of a single net are presented. The emphasis in this chapter is on the mechanism for the periodic exchanging of state information and the way this information is used for the hybrid access control. Global synchronization and time stamping are incorporated into the system and presented in Chapter 6. The global event synchronization mechanism, which is imposed on the system, also uses the implicit and explicit time information, which is exchanged periodically by the control messages. All events are time stamped by a synchronous system clock, thereby preserving total event ordering.

The next step is combining the nets into an optical hypergraph network, which is described in Chapter 7, with emphasis on two basic configurations: two–dimensional regular hypergraphs and two–dimensional partial hypergraphs. The fault tolerance enhancement for the system is described in Chapter 8. In particular, it is shown how the loss of packets can be prevented, based again on the periodic exchange of state information. Chapter 9 shows how real–time (voice) communication is integrated into the system. The voice integration is an immediate outcome of the global synchronization. Final discussion, conclusions and further considerations are presented in Chapter 10.

1.4. Optical Architecture for Large Area Network

A metropolitan area network is supposed to be significantly larger than today's local area networks (e.g., Ethernet) with respect to the following parameters: (i) 10–100 times the number of nodes, (ii) 10–100 times the area, (iii) 10–100 times the medium bandwidth, and (iv) capability to integrate several types of communication.

In general, an optical architecture is comprised of three parts: (i) the connectivity among the network nodes (i.e., the topology), (ii) the explicit physical properties of the network (e.g., transmission method, time partitioning, baud rate), and (iii) basic principles of operation, such as access control, routing, and buffer management. Many of the network examples which are found in the literature concentrate on only one of these three aspects, and often it is not clear how and how well the other architectural aspects are solved.

When examining the scope of optical architecture, the following classes can be identified:

(1) Centralized point–to–point architecture – similar to the telephone network. In this architecture high bandwidth fibers replace wires. This architecture is based on a centralized switch, which can be a major bottleneck. The major advantage of the centralized switch architecture is that it can be controlled more in a simple manner than a distributed architecture. A major disadvantage is that it is harder to extend this architecture. Two recent examples for this approach can be found in [LMHo86] and [VlLe84]. Clearly, the switch architecture is a possible solution, which is relatively well understood and simple to implement.

(2) Optical local area network (LAN) – this architecture is limited in its size and is constructed of a single shared medium (a ring or a bus), with possible gateways to other networks. A typical example is Fibernet II: A Fiber Optic Ethernet [SRNJB83], or Hubnet ([LeBo83] and [LeBoIk84]), D–NET [TCJ83], or active star LAN [Kama87].

(3) Backbone optical networks – which have been proposed for metropolitan area networks. These networks are characterized by having dual rings operating in opposite directions. Typical examples are FDDI [JoIy84] and MAGNET [Lazar85]. In some configurations several of these dual rings are connected together [Sze85].

(4) Distributed networks – a network consisting of several nets, where each net is a shared medium or a bus. The switching of messages is done in a distributed manner. Most of these designs are top–down without explicitly showing how they are realized and operated. Typical examples are found in [Witt81], [Ullm84], and [RePa85].

The proposed optical hypergraph is a distributed network, as in the last example. The major differences are: (i) the design methodology is bottom–up, trying to show how all major aspects of the system can be realized, (ii) the design is based on some specific optical properties (single mode fibers, passive optical couplers), and (iii) the design tries to make an efficient use of the high bandwidth (about 1 gigabit/sec), utilizing different design principles (e.g., periodic exchange of state information).

CHAPTER 2.

SYSTEM'S PRINCIPLES

In this chapter the basic principles for constructing the system are presented, and its principles of operations are discussed. The concepts which will be defined and developed in this chapter will be used throughout the thesis.

2.1. The Scope of the System

The system, in the context of this research, is defined as the network topology plus the interface functions, as shown in Figure 2.1. The shared buses of the hosts are the boundaries of the system. Each host bus may be shared by several computing units. In general, the units which reside beyond the host bus are beyond the scope of this thesis. From the host point of view, the network is a resource, like other memory resources. Via the shared host bus, a computing unit writes or reads a message to or from a buffer in the network's interface.

A node is defined as a point in space from which there is an access to the network, via a host bus. In the following discussion the term "node" refers to the interface functions at some point.

2.2. The Network Topology

The network topology is a hypergraph, which is formally defined as [Berg73]: Let $X = \{x_1, x_2, \cdots , x_p\}$ be a finite set of nodes, and let $F = \{E_i \mid i \in I\}$ be a family of

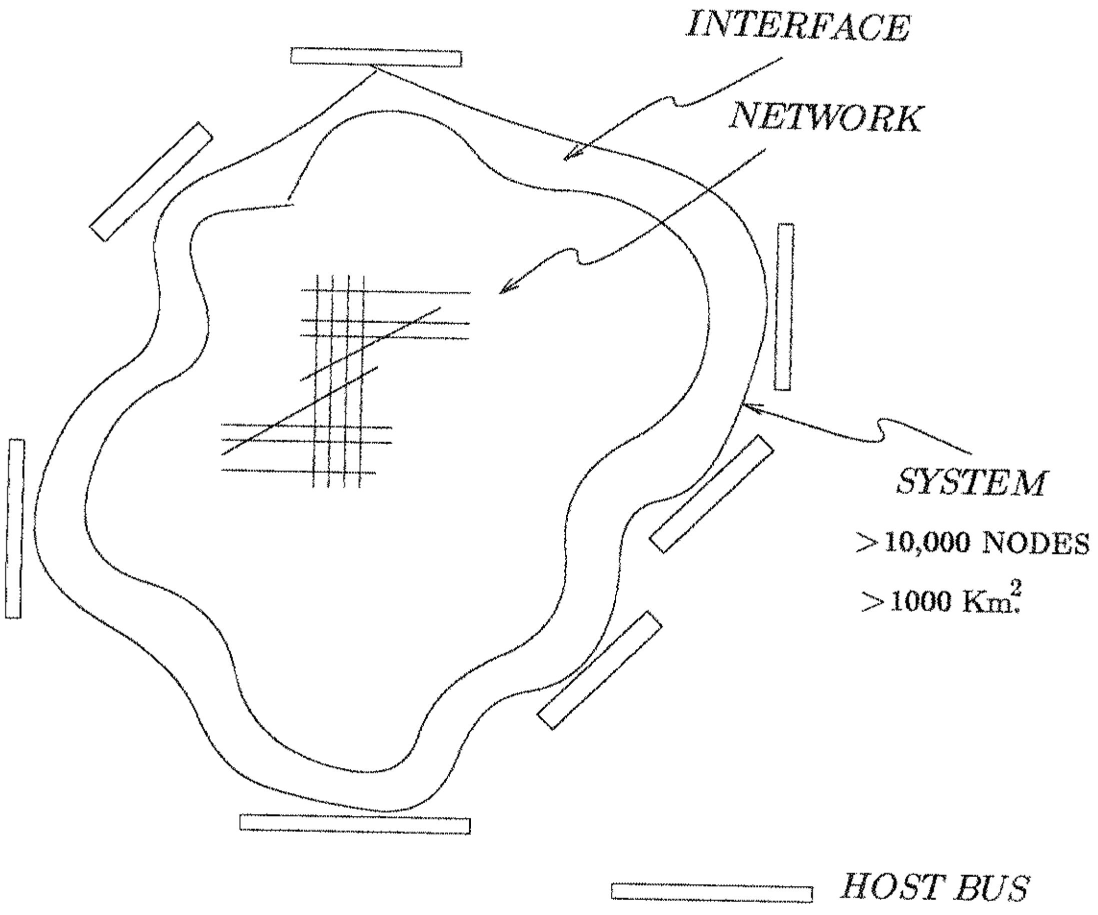


Figure 2.1: The System Description

subsets of X. The family F is said to be a hypergraph on X if (1) $E_i \neq \phi$, and (2) the union of all E_i is equal to X. The couple $E = (X,F)$ is called a *hypergraph.* The elements $x_1, x_2, \cdots, x_p$ are called the *vertices.* The sets $E_1, E_2, \cdots, E_m$ are called the *edges of the hypergraph.*

In this research four different hypergraph configurations were considered.

(1) Two–dimensional regular hypergraph (2D–R) – each edge (net) E_i has exactly n vertices (nodes), such that $n^2 = p$ (p is the total number of vertices in the hyper-

graph), and each vertex belongs exactly to two edges (E_i). Figure 2.2a is an example of a 4–by–4 regular hypergraph in the $x-y$ plane; each edge has exactly 4 vertices and the total number of vertices is 16. Figure 2.2b is a representation of a 3–by–3 2D–R hypergraph, such that each net is realized by a centralized optical star.

(2) Two–dimensional, partial hypergraph (2D–P) – the set of p vertices is divided into $2k$ subsets. Each E_i has exactly n vertices, such that (i) k of its vertices belong to exactly two edges, (ii) a vertices belong only to one edge, and (iii) $k+a=n$. The total number of vertices in the partial network is

$$p = k^2 + 2ka = n^2 - a^2$$

Described in Figure 2.3 the different possible layouts of 2D–P. Note that since

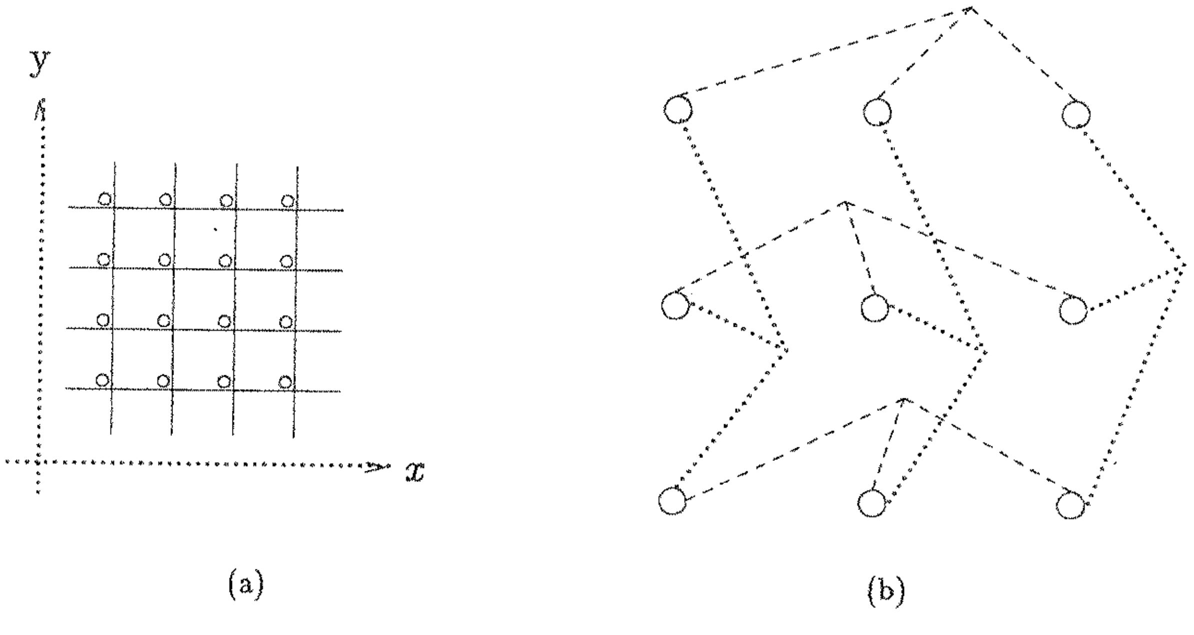


Figure 2.2: Two–Dimensional Regular Hypergraph

every edge is a centralized optical star, the nodes (vertices) are **indistinguishable** with respect to the star's center. Therefore, the a vertices of each net, which belong only to one edge, can always be rearranged as shown in Figure 2.3b or Figure 2.3c.

(3) Three–dimensional, regular hypergraph (3D–R) – each edge (net) E_i has exactly n vertices (nodes), such that $n^3 = p$ (p is the total number of vertices in the hypergraph), and each vertex belongs exactly to three edges (E_i). Figure 2.4 is an example of a 3–by–3–by–3 regular hypergraph in $x-y-z$ space; each edge has exactly 3 vertices, and the total number of vertices is 27.

(4) Three–dimensional, partial hypergraph (3D–P) – the set of p vertices is divided into $3k^2$ subsets. Each E_i has exactly n vertices, such that (i) k of its vertices

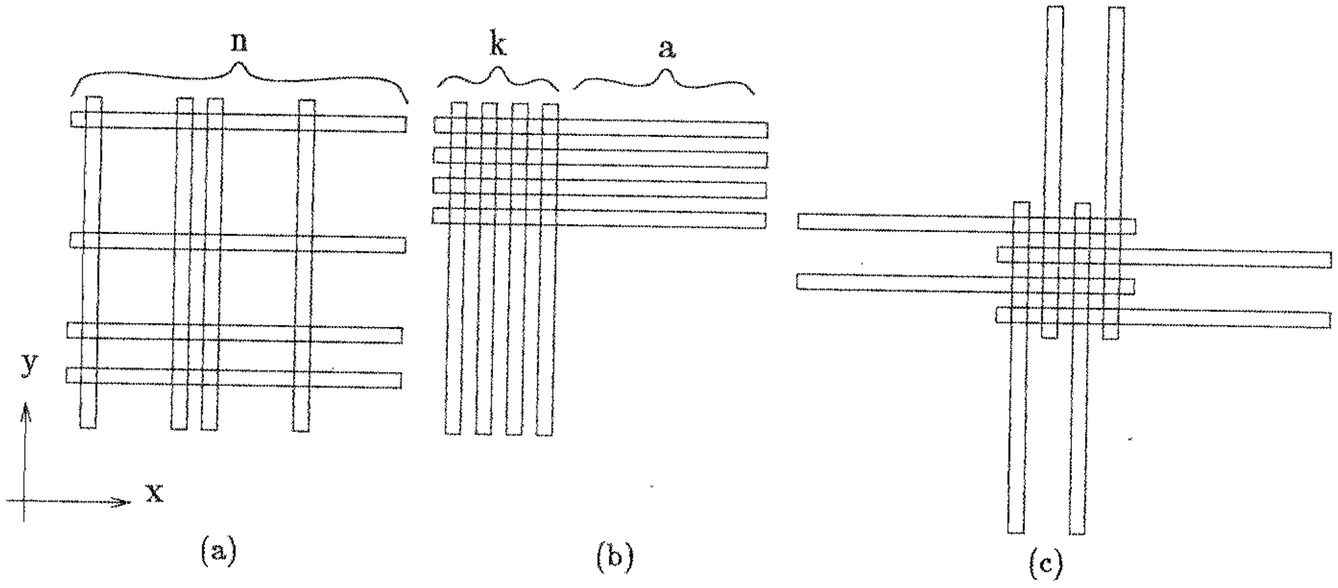


Figure 2.3: 2D–2P Partial Hypergraph

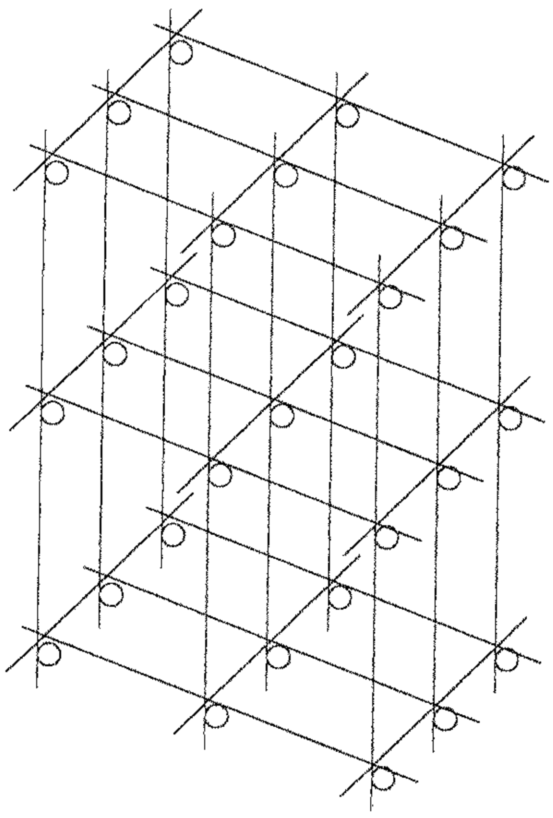


Figure 2.4: Three–Dimensional Hypergraph

belong to exactly three edges, (ii) a vertices belong only to one edge, and (iii) $k+a=n$. The total number of vertices in the 3D–P network is

$$p = k^3 + 3k^2a = n^3 - a^3 - 3a^2k$$

2.3. The Optical Star

Each net is a passive optical star, which is realized by centralized passive optical couplers, as shown in Figure 2.5, which is an array of single mode, optical couplers. A general discussion on star couplers is found in [Marh84]. It is assumed that the optical star is built in a very small area in space, so the communication on the net merges to one point in space (called the center of the star), and then broadcasts back to all the nodes of the net. The optical hypergraph is composed of multiple optical stars, as shown in Figure 2.2b, where there are 9 nodes and 6 optical stars, with each node having access to two optical stars via its two ports.

The single-mode, optical coupler is a device with two input terminals (C^A and C^B), and two output terminals (C^C and C^D).

Let $G(A,D)$ – be the transmittance from point A to point D, representing the fraction of power from A which reaches D.

The optical coupler has the following reciprocity property:

$$G(C^A,C^D) = G(C^B,C^C)$$

$$G(C^A,C^C) = G(C^B,C^D)$$

The major consequence of this property is that it is impossible to send from both inputs more than half of the energy to only one of the outputs. For example, if

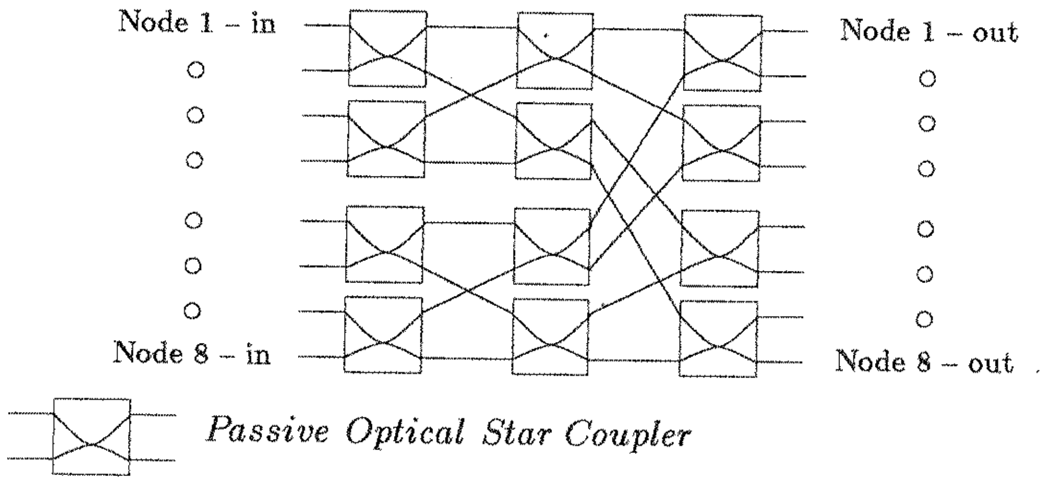


Figure 2.5: Optical Star

$G(C^A,C^D)=\frac{3}{4}$, then $G(C^A,C^C)=\frac{1}{4}$, and hence, $G(C^B,C^D)=\frac{1}{4}$. Therefore, for practical reasons it can be assumed that the transmittance of the optical coupler is exactly one half, i.e., the coupler attenuates the signal by $3db$.

In [NTM85], Tobagi et al. analyze different net configurations which use passive optical couplers. Their results demonstrate the feasibility of constructing a passive fiber optic net with about 100 nodes, within an area of more 1000 square kilometers.

The net is constructed of single–mode fibers with very low losses, and $3db$ optical couplers (only symmetric couplers are considered). The following is a comparison among three passive topologies: star, binary–tree and ring (or linear bus).
Given that (a) the transmitter emits a unit of energy into the net, (b) the net has N nodes (N is power of 2), and (c) the only losses are by splitting the energy by the optical couplers. *What is the minimum energy that each node receives?*

(1) Star topology –

$$E_r=\frac{1}{2^{\log_2 N}}=\frac{1}{N}$$

(2) Tree topology –

$$E_r=\frac{1}{2^{(2\log_2 N)-1}}=\frac{2}{N^2}$$

(3) Ring or linear bus topology –

$$E_r=\frac{1}{2^N}$$

Clearly, the passive star topology distributes the energy most efficiently. Detailed discussion of the energy issue, with respect to all losses, can be found in [NTM85].

2.4. Time Partitioning

The timing relationship among the system's nodes and nets has a very important role in the implementation of high–level system functions, e.g., distributed operating systems and voice communication.

There are three main reasons for time slotting:

(1) To be able to transmit a single bit of information into the centralized net, almost every bit period. Thus, the upper bound on the net utilization is almost 100%. Note that as the bandwidth gets higher the size of the packet in the serial link gets shorter. In a large area network with a bandwidth over one gigabit/sec it is reasonable to assume that several packets can be in the channel at the same time. Without slotting, the next one to transmit must wait until it sees the tail of the current packet. Thus, as the bandwidth gets higher more communication capacity is wasted.

(2) To be able to broadcast multiple short control messages successively from multiple nodes.

(3) To support a global system clock, which maintains global event synchronization of all the system's nodes, and enables time stamping of all communication and computation events.

(4) To have a well–defined global state (or instantaneous description), and well–defined global state transitions.

(5) To reduce the probability of collision, as in the case of slotted Aloha versus pure Aloha [Abra77].

The system is synchronized by each time slot. Each node's interface has a slot counter for measuring the time. All the system's slot counters are synchronized and have the same reading (in Chapter 6 there is a detailed description and analysis of the mechanism for achieving global event synchronization).

The slots are grouped into frames with f slots per frame. The duration of one time slot and the value of f are determined such that one time frame duration is larger than the maximum delay of any node to the center of the net and back. The duration of a slot is measured in bit periods, and is about the size of a typical packet or page in the system.

2.4.1. Periodic exchange of state information

Each slot is subdivided into $r+1$ minislots. The first r minislots are very short control minislots (CMSs), and the last one is a data minislot (DMS). The minislots are used by different nodes, so during each time slot $r+1$ different nodes are transmitting over the net.

The main reason for dividing the slot into control and data parts is for functional and physical partition between the communication (or computation) controls, and the actual data transfers (or processing).

2.4.2. Parcels for voice communication

During the DMS, which occupies most of the time slot, one packet of data is sent from one node. The data packet may be subdivided into p parcels. Each parcel may have a different destination or destinations over the net. The use of parcels enables the node to mix small and large data messages in an arbitrary fashion. This makes the communication over the net flexible and dynamically adaptable to different types of communication. As a result, the use of the communication resource of the net becomes more efficient. The use of parcels, together with global event synchronization, enables the simple integration of voice and data communication (see Chapter 9).

2.5. Basic Principles of Operation

The operation of this system follows some general principles that are defined in this section.

2.5.1. Global synchronization and total event ordering

The time on each of the system's nets is partitioned in the same way, and with equal slot duration. Global synchronization means that the time patterns on all nets cannot differ by more than half a time slot (see Chapter 6). Each node interface has a slot counter (local event clock). All the system's slot counters are synchronized, such that at the end of each slot all the system's slot counters have the same reading for a period of time which is greater than zero. With this value, the current events at each node are time stamped. Thus, if all the events in the system are time stamped, then the system preserves total temporal event ordering.

2.5.2. Synchronous and asynchronous operation

There are two possible views on this system, the microview and the macroview. In the microview, the system is asynchronous, i.e., every node has its own local clock, which is used for transmission and for the interface's finite state controllers. In the macroview, the system is synchronous, and its basic event is a single time slot. The synchronous event is three or four orders of magnitude longer than the asynchronous event, a single bit period.

2.5.3. Open mode operation

Global event synchronization and total event ordering enable the system to perform distributed computing functions in an *open mode* fashion. For example, a process can send a request with a time stamp to all the other cooperating processes. The request will be granted, without an explicit acknowledge after the two conditions are true: (i) the process waits the necessary required time for the request to reach all the other cooperating processes, and (ii) the requesting process does not receive similar requests, with an earlier time stamp, during its waiting period.

2.5.4. State information and common knowledge

Control and state information are transmitted during the control minislots (CMSs), so that all nodes periodically exchange state information. This information is considered **local common knowledge** after it has reached all the net's nodes. Once this information has reached all the **system's** nodes, it is then considered **global common knowledge.**

It is not required that **common knowledge** information be kept explicitly in the node. The interface can perform real–time computation on the information, incorporate the result n some local parameters, and discard the original data. So, the interface does not contain a large number of replicated parameters.

For example, the state information, which is exchanged periodically, can be used to integrate, in a uniform manner, the following functions:

(i) Data Communication

(ii) Distributed Computation

(iii) Distributed Data Base Management

(iv) Real–Time Voice Communication

2.5.5. Round–robin scheduling

Round–robin scheduling is very simple to implement in real–time, and can guarantee fairness. Moreover, for a homogeneous system, round–robin scheduling is efficient. In principle, other scheduling algorithms can be implemented, but for the presentation, understanding, and analysis round–robin is simpler. An important condition for implementing any other scheduling scheme is to be able to execute it under real–time constraints, which become tighter as the net's bandwidth increases.

CHAPTER 3.

CONSERVATIVE CODE FOR MULTIPLE ACCESS CHANNELS

3.1. General

The family of conservative codes is a new coding scheme for preserving the time integrity of serial communication over a multiple access channel. These codes are characterized by a constant number of transitions in each codeword and a known delimiting transition (rising or falling edge) at the end of each codeword ([Ofek87a], [Ofek87b]).

The conservative code enables a receiver and its serial–to–parallel converter to operate without a training period (preamble), i.e., no phase–locked loop is used for clock recovery. Instead, very accurate delay lines are used at the receiving side for decoding and for serial–to–parallel conversion. At very high bandwidth, delay lines can be accurately realized by using transmission lines or wave guides.

The main objective of this code is to preserve timing information, as opposed to codes which preserve data integrity. In fact, these two coding objectives can be combined by a two–level transformation, as shown in Figure 3.1. At the transmitter side, the error–detecting/correcting transformation is first applied, and then, the conservative code is used for adding the timing information. At the receiver side, the conservative code is used for the decoding and parallel–to–serial conversion, and then, the error–detecting/correcting function is applied. These two levels of encoding/decoding correspond to the first two levels of the communication protocol: the conservative transformation is part of the physical level interface, while the error–detecting/correcting codes are part of the link level protocol.

In most existing designs the receiver's clock is explicitly recovered from the serial codewords with a phase–locked loop. This clock is then used for decoding and serial–to–parallel conversion. Hence, the major requirements of the encoding/decoding scheme are to ensure a sufficient number of transitions. Common examples for this kind of code are Manchester, biphase, 4B/5B, and others (see [WiFr83], [Sore84], [Seve80], [Mang83], [Lacr84] and [JoIy84]). None of the existing coding schemes have a fixed ratio of total number of transitions to the total number of bits. Therefore, it is more difficult to decode and to repartition the serial information into bytes and words without a phase–locked loop.

Two additional constraints are imposed on the conservative encoding scheme: (i) balancing each codeword, i.e., making the number of zeros and ones equal, and (ii) limiting the maximum run–lengths at the high and low levels, i.e., the maximum level duration is limited. These two restrictions limit the DC shift of the receiver, and thereby

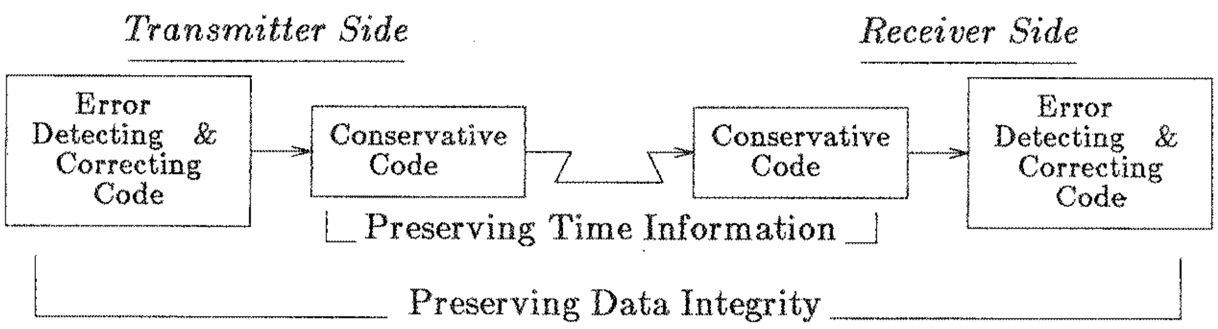

Transmitter Side
Receiver Side
Error Detecting & Correcting Code
Conservative Code
Conservative Code
Error Detecting & Correcting Code
Preserving Time Information
Preserving Data Integrity

Figure 3.1: The Encoding/Decoding Protocols Levels

increase the decoding reliability and decrease the receiver complexity. It is shown that both constraints can be imposed simultaneously without a significant degradation in encoding efficiency.

The family of conservative codes is a feasible alternative to other known codes. Applying the additional constraints of balancing and limited run–length helps to simplify and improve the encoder and decoder design. The balancing constraint reduces the number of variables of the encoding/decoding Boolean functions from n to $n/2$. The maximum run–length constraint can reduce the time delay needed for realizing the decoding and encoding of Boolean functions.

3.2. Description of the Conservative Code

The family of conservative codes is defined on a block of n bits (let n be even) $a_1, a_2, \ldots, a_n$, with exactly $n/2$ transitions in every codeword. If a_i can be either -1 or $+1$, then the encoding is defined by

$$\sum_{i=1}^{i=n-1} a_i a_{i+1} = +1$$

and

$$a_n a_{n+1} = -1, \;\; or \;\; a_n = 1 \text{ and } a_{n+1} = a_0 = -1$$

The a_{n+1} is the first bit (a_0) of the next codeword. In the more general case, for b transitions per codeword, $b \leq n$, and b is even,

$$\sum_{i=1}^{i=n-1} a_i a_{i+1} = n - 2b + 1$$

and the known delimiter always occurs at the end of the n^{th} bit–cell of each codeword. If b is odd, then every other codeword is complemented bit–by–bit, and the above equation holds. The transitions are used at the receiver side to count how many bits have arrived and to clock them in parallel into an array of shift registers, with the codeword's delimiting transitions.

The conservative encoding scheme maps l-bit data words onto n-bit codewords. For n even, b maximizes the different possible codewords when $b = \frac{n}{2}$. Given n and l, the number of possible different conservative codewords is computed as follows: each codeword c can be mapped uniquely to c' such that if there is a transition at the end of the bit cell, then this bit is mapped to one, and if there is no transition at the end of the bit cell this bit is mapped to zero. So, the mapped codeword c' has exactly $b-1$ ones which are arbitrarily placed in c', and the b^{th} one is always after the n^{th} bit cell. Thus, the number of different codewords is $\binom{n-1}{b-1}$, and the values of l, n and b satisfy the following inequality:

$$2^l < \frac{(n-1)!}{(n-b)!(b-1)!}$$

The efficiency of the encoding, μ, is the ratio of l to n.

If n is exactly divisible by four, then the codeword's bit counter need use only the rising edges or the falling edges. In this case, the counter counts to $\frac{n}{4}$ in order to determine the delimiting transition of a codeword. If n is not exactly divisible by four, the delimiting transition must be determined with a counter that uses both edges, which is more complex and can reduce the digital electronic bandwidth. In the following discussion it is assumed that $n = 4k$.

3.2.1. The 1–2 PWM code

The 1–2 pulse–width–modulation (PWM) code is a special case of the conservative scheme, with exactly one transition for every bit. In this scheme, zero is encoded as a level for one clock period, and one is encoded as a level for two clock periods, as shown in Figure 3.2. The decoding circuitry is very simple, as shown in Figure 3.3. One shift register is used for recording the odd bits, and the other shift register for recording the even bits. The outputs of the two registers are combined into one parallel word. Since the transmission length depends on the actual data, and not only on the number of bits in each packet, the baud rate is not constant. The efficiency of this encoding averages 67%, and in worst case is only 50%. If the transmission is slotted (TDMA), then one must assume the worst case efficiency of 50%.

3.2.2. The 8B/12B conservative code

Another example is encoding 8 bits of data into a 12–bit codeword (8B/12B conservative scheme). Every codeword has (i) 6 transitions, (ii) the 12^{th} bit is always one, and (iii) the first bit is always zero. Hence, every codeword has a falling edge as its delimiter. The number of bits is increased by 50%, and the encoding efficiency is 67%.

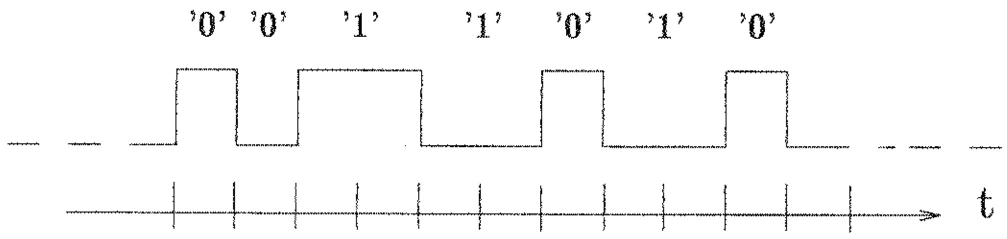


Figure 3.2: 1–2 PWM Encoding Scheme

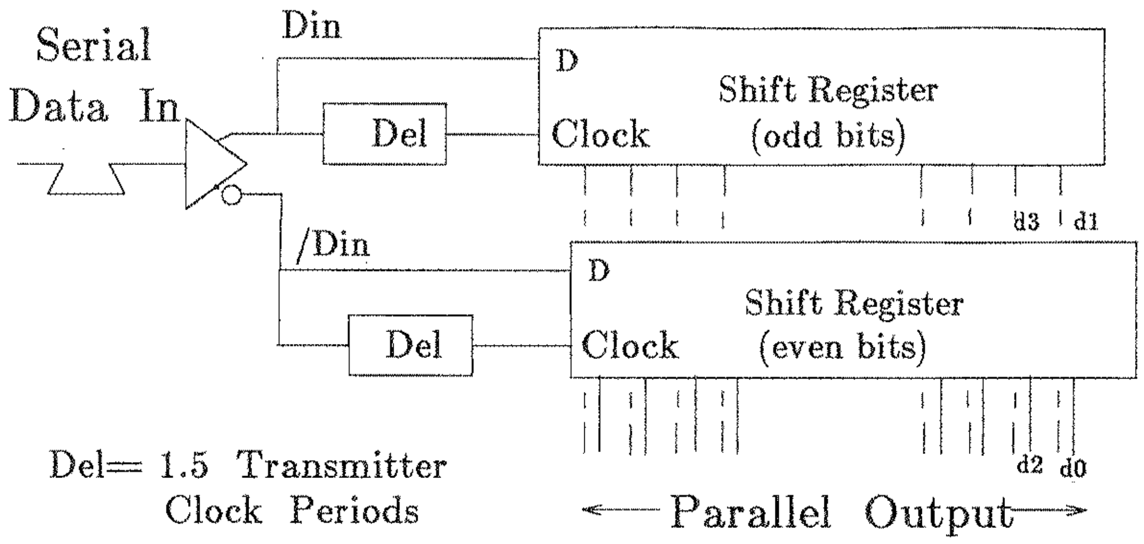

Serial Data In
Din
/Din
Del
Del
D
Clock
Shift Register (odd bits)
D
Clock
Shift Register (even bits)
d3
d1
d2
d0
Del= 1.5 Transmitter Clock Periods
Parallel Output

Figure 3.3: The 1–2 PWM Decoder

Eight bits of data need 256 different 12–bit codewords. The total number of different possible codewords is $\binom{11}{5} = \frac{11!}{5!6!} = 462$, which is greater than 256.

Higher–order encoding is possible, e.g., 12–bit data into a 16–bit code (12B/16B conservative scheme), or 16–bit data into a 20–bit code (16B/20B conservative scheme), with respective efficiencies of 75% and 80%.

3.3. Analysis of the Conservative Code

For practical purposes, two additional constraints are imposed on the encoding. First, limited run–length, i.e., the maximum duration of the serial signal at a high or low level, is limited to m bits. Second, the codeword is balanced, such that each codeword has the same number of low and high bits.

The main use of the conservative encoding scheme is in very high bandwidth fiber optic networks. Since fiber optic receivers are usually ac–coupled, using capacitors, an excessive dc level shift can cause errors in the interpretation of serial data. Also, the clock information encoded into the serial data must be accurately interpreted. The above constraints limit the dc shift and, therefore, increase the decoding reliability and decrease the optical receiver complexity. It will be shown that both constraints can be simultaneously imposed without a significant efficiency degradation.

Balanced block encoding schemes have been proposed by [WiFr83], [Knut86] and [Leis84]. Some other almost balanced codes are described by [JoIy84], [Mang83], [Lacr84]. But none of these attempts to conserve the number of transitions in each codeword. Limited run–length coding is treated in [Fran70], [Fran82] and [HoOs75].

3.3.1. Analysis

The analysis proceeds in two steps: first, the total number of different conservative codewords of length n under the above constraints is computed, then the efficiency is computed.

Figure 3.4 shows the model for computing the number of different codewords, as a square wave with $2k$–transitions ($b = 2k$), i.e., k bits of high level and k bits of low level. All the high and low levels can be extended by a total of $2k$ bit periods. If e_i is the amount each high or low level is extended, then

$$e_1 + e_2 + ... + e_{2k} = 2k,$$

with $e_i \in O, 1, 2, \cdots , 2k$.

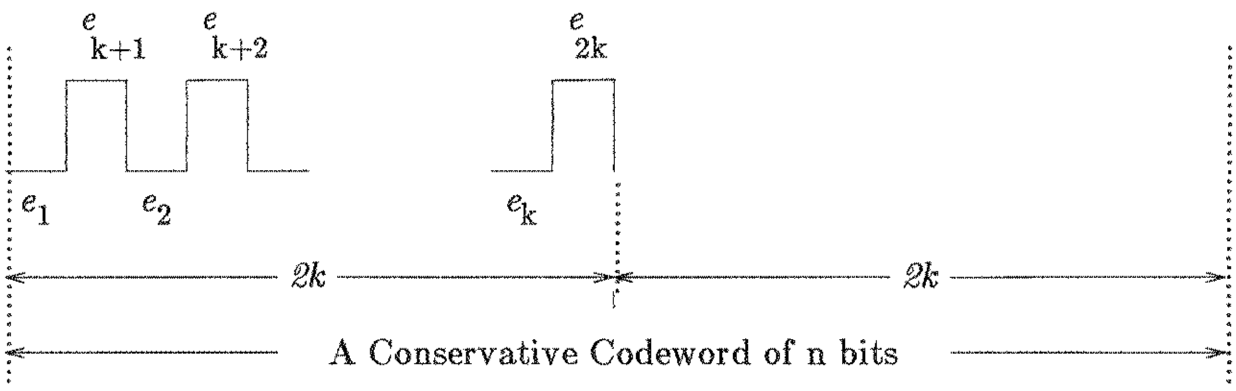


Figure 3.4: The Analysis Model

Thus, with no additional constraints, the total number of different conservative codewords is equal to the number of different integer solutions of the above equation:

$$C(n) = \binom{4k-1}{2k-1} = \binom{n-1}{\frac{n}{2}-1} = \frac{(n-1)!}{(2k-1)!2k!}$$

3.3.2. Conservative/balanced encoding

Under the balancing constraints, the high levels and the low levels are independently extended by k bit periods independently. Thus, the number of different low–level duration arrangements is equal to the number of different integer solutions of the following equation:

$$e_1+e_2+e_3+\ldots+e_k=k$$

with $e_i \in 0, 1, 2, \cdots, k$, which is:

$$CB'(n) = \binom{2k-1}{k-1}$$

Since the high and low levels can be arranged independently, the total number of possible arrangements of a conservative/balanced codeword of length n is

$$CB(n) = (CB'(n))^2 = \binom{2k-1}{k-1}^2 = \left[\frac{(2k-1)!}{(k-1)!k!}\right]^2$$

Figure 3.5 describes the encoding efficiency $\mu(n)$ as a function of n, for the two cases of conservative and conservative/balanced codes.

$$\mu_C(n) = 100\frac{\lfloor \log_2 C(n) \rfloor}{n}$$

$$\mu_{CB}(n) = 100\frac{\lfloor \log_2 CB(n) \rfloor}{n}$$

It is apparent that the efficiency of the conservative/balanced encoding is about 80% for codewords larger than 32 bits. Since the high– and low–level encodings are independent, it is possible to realize the encoder by using two look–up tables of $n/2$ address lines ($2^{n/2}$ entries); while the realization of a conservative encoding with a look–up table requires n address lines (2^n entries). Therefore, if the maximum table size which is reasonable to construct has 16 address lines, the efficiency of the conservative/balanced code is 78%, and is better than the 75% efficiency of the conservative encoding.

3.3.3. Encoding with a limited run–length

This section analyzes the efficiency of the conservative code with limited run–length of m at a high or a low level. The problem can be expressed as follows: given the above square–wave form with $2k$ transitions, all the high and low levels are extended by a total of $2k$ bit periods not exceeding $m-1$ at each continuous level. If e_i is the amount each high or low level is extended ($e_i \in \{0, 1, 2, .. , m-1\}$, $m-1 \leq 2k$), then the number of different arrangements is equal to the number of different integer solutions of the following equation:

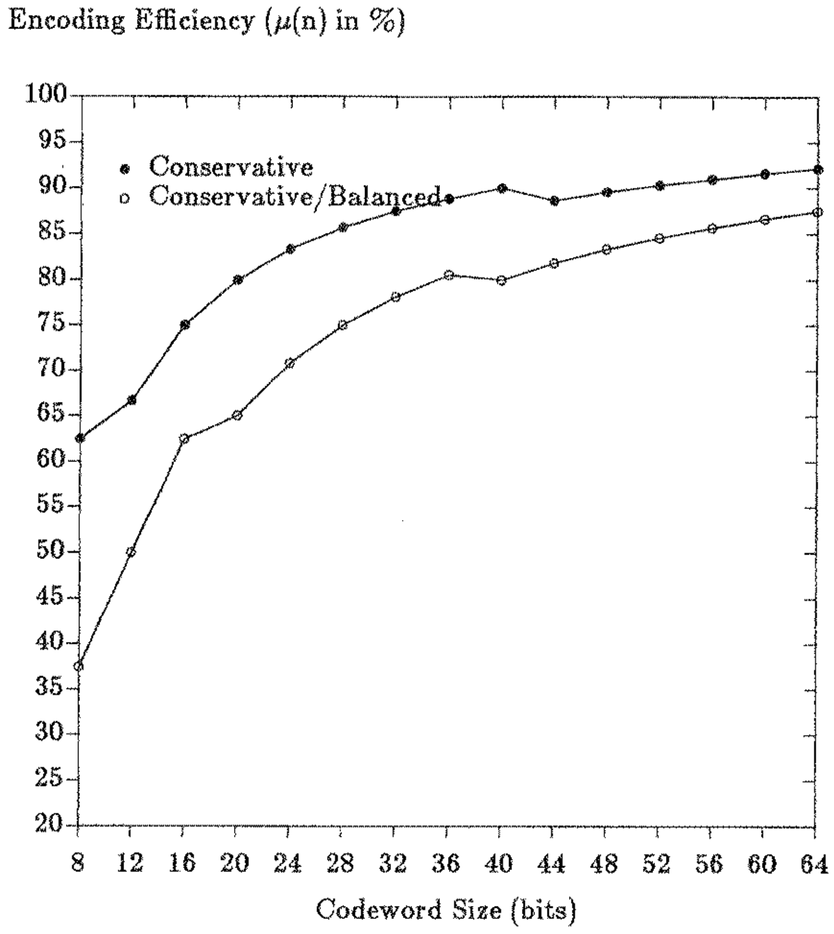


Figure 3.5: Conservative and Conservative/Balanced

$$e_1+e_2+e_3+...+e_{2k} = 2k$$

which is the coefficient of x^{2k} in the polynomial expansion of the following generating function:

$$h(x)=(1+x+x^2+x^3+..+x^{m-1})^{2k}$$

The coefficient is expressed as the function *CR(n,m)*:

$$CR(n,m) = \sum_{i=0}^{i \le 2k/m} (-1)^i \binom{2k}{i} \binom{n-1-mi}{2k-mi}$$

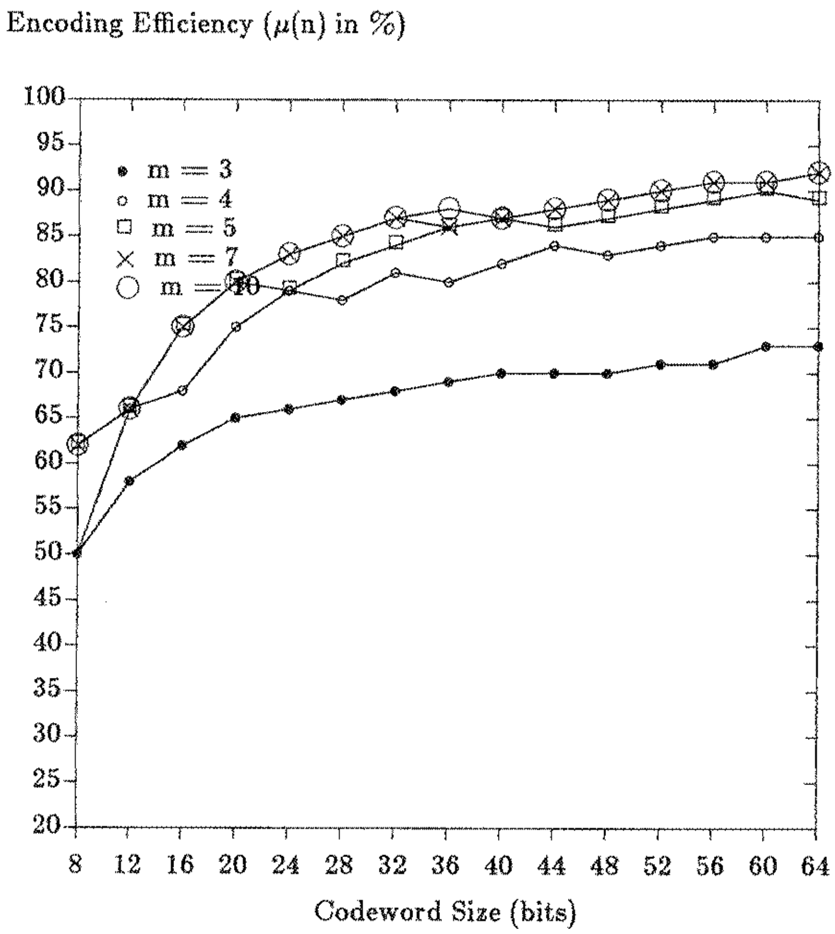


Figure 3.6: Encoding with a Limited Run–length m

Figure 3.6 shows the encoding efficiency $\mu(n)$ as a function of n, for conservative codes with a limited run–length of m.

$$\mu_{CR}(n)=100\frac{\left\lfloor \log_2 CR(n,m) \right\rfloor}{n}$$

From Figure 3.6 it is apparent that the encoding efficiency is little improved for a run–length greater than 4.

3.3.4. Balanced encoding with limited run–length

In this section both constraints are imposed on the conservative code. Formally, the code has the following constraints: (i) $n/2$ transitions, (ii) $n/2$ ones, (iii) a falling edge as its delimiter, and (iv) maximum run–length at the high or low level of m.

This counting problem can be solved independently for the high and low levels. The number of different arrangements for the high (or low) level is the coefficient of x^k in the polynomial expansion of the following generating function $(m-1 \leq k)$:

$$h(x) = (1+x+x^2+x^3+..+x^{m-1})^k$$

Raising this coefficient to the power 2 will produce the total number of possible arrangements, which is expressed in the function $CBR(n,m)$:

$$CBR(n,m) = \left[\sum_{i=0}^{i \leq k/m} (-1)^i \binom{k}{i} \binom{2k-1-mi}{k-mi} \right]^2$$

Figure 3.7 describes the encoding efficiency $\mu(n)$ as the function of n for conservative/balanced codes with limited run–length of m.

$$\mu_{CBR}(n) = 100 \frac{\left\lfloor \log_2 CBR(n,m) \right\rfloor}{n}$$

Again, the encoding efficiency is little improved for a run–length greater than 4, and that applying both constraints changes the encoding efficiency very little.

3.3.5. Collision detection of the conservative/balanced code

The star topology of the net guarantees that if a collision has occurred, it can be seen by all the net's nodes for **any packet length**. The following design enables the nodes to detect and diagnose collisions, by encoding the data with the

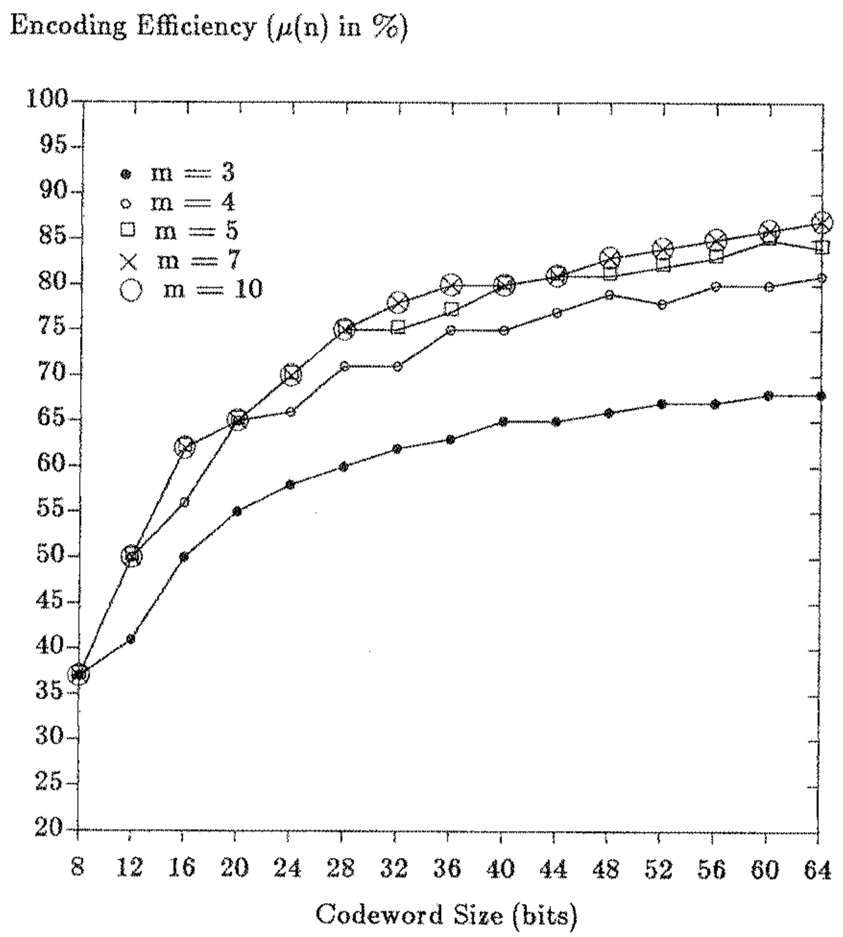


Figure 3.7: Conservative/Balanced with Run–length m

conservative/balanced code. Thus, each codeword has a fixed number of transitions, and an equal number of zeros and ones. After an optical collision the high–level duration is longer and the low–level duration is shorter. Hence, the codeword is not balanced anymore. A simple combinational network can detect the nonbalanced codeword, as shown in Figure 3.8, and then indicate a collision or an error. The error–

detection circuit counts the number of ones in a codeword of n bits. If the number of ones is larger than $\frac{n}{2}$, then it is probable that a collision has occurred. If the number of ones is less than $\frac{n}{2}$, then the failure is probably due to other causes. If the number of ones is exactly $\frac{n}{2}$, then it is a legal codeword. This simple error–detection mechanism is very effective, and uses the inherent properties of the conservative/balanced encoding scheme.

3.4. Generating the Look–up Table for the Code

In this section simple methods for generating the look–up table for encoding and decoding under the various constraints are presented.

3.4.1. Conservative code with no constraints

The transformations of the l–bit data word into an n–bit codeword and back can be done by using one look–up table for the encoder and one for the decoder. These tables are realized by two combinational or sequential networks, i.e., ROM or finite state machine.

Let U be the data word, $U = (u_1, u_2, \cdots, u_l)$, and C be the codeword, $C = (c_1, c_2, \cdots, c_n)$. Let b be the number of transitions in every codeword.

First, all possible binary data words of length l are listed in their binary order. The size of this list is 2^l. As data words we select a subset of the above list, since not all possible may be used, it is assumed that the size of data word subset is $L_{DATA} \leq 2^l$.

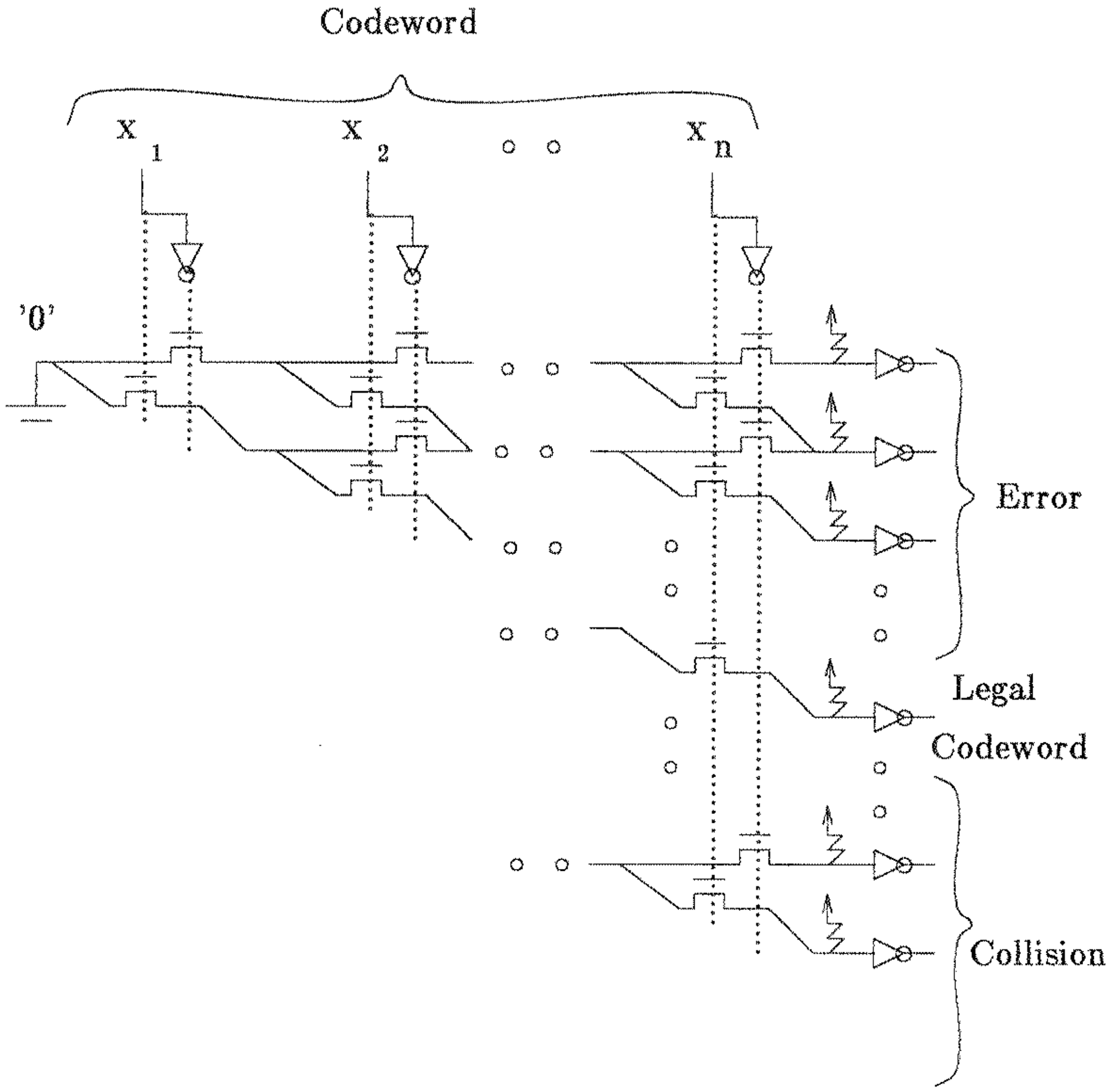


Figure 3.8: Collision and Error Detection for Conservative/Balanced Code

The list of all possible conservative codewords is generated in the following steps: Step 1 – a line of length n is divided into n segments. There are $n-1$ possible positions for $b-1$ transitions, i.e., a transition at the end of the first line segment, or 2^{nd} segment, and so on. The last transition that can be selected is at the end of the $n-1^{th}$ line segment. The b^{th} transition is the delimiter, and is always at the end of the n^{th}

line segment. Thus, $b-1$ transitions are selected from $n-1$ positions in the codeword. Let $\{1, 2, 3, \ldots, n-1\}$ be the set of different transition positions in the codeword. Then all possible subsets of length $b-1$ are listed in a **lexicographical order**. There are $\binom{n-1}{b-1}$ different subsets.

Step 2 – from the above list of subsets, the list of all different codewords is generated. Using the following procedure each subset $\{s_j, 1 \leq j \leq b-1\}$ is mapped uniquely into a conservative codeword.

Note that if, the number of transitions (b) is even, then there are two equivalent cases: (i) the first bit is always zero, and (ii) the first bit is always one. If b is odd then the first bit alternates between zero and one. The following procedure generates codewords such that the first bit is always zero. If the first bit would be one, then a complement list of conservative codewords would be obtained.

```
$v = 0$
$c_1 = 0$
DO $i = 2$ to $n$
    IF $i-1 \in \{s_1, .., s_{b-1}\}$,
        THEN $v = v+1 \bmod 2$
    $c_i = v$
END
```

Note that if b is odd, then, while transmitting a sequence of codewords, all even codewords should be inverted (or complemented) bit–by–bit. Thus, the delimiting transition would be preserved. At the receiving side, after the serial–to–parallel conversion, these codewords should be inverted back to the original codeword.

In general, a subset of codewords is selected from this list, with a size of $L_{CON} \leq \binom{n-1}{b-1}$. Clearly, for one–to–one transformations from the subset of data words to the subset of codewords, $L_{DATA} \leq L_{CON} = p$.
The encoding table is a one–to–one mapping from the data subset to the codeword subset; the decoding table is the reverse. All the unused codewords in the codeword list may be used for an error signal or for control.

Note that the one–to–one mapping is arbitrary, and there are $p!$ different possible mappings.

In the following example $l=4$, $n=7$, and $b=4$, and a possible one–to–one mapping is

Data Word	Codeword
0000	0101111
0001	0100111
0010	0100011
0011	0100001
0100	0110111
0101	0110011
0110	0110001
0111	0111011
1000	0111001
1001	0111101
1010	0010111
1011	0010011
1100	0010001
1101	0011011
1110	0011001
1111	0011101
Unused	0001011
Unused	0001001
Unused	0001101
Unused	0000101

The unused codewords may be used for error detection or communication control.

3.4.2. Conservative balanced code

Generating the mapping table for the conservative balanced code is done as described before, with an additional step that eliminates the nonbalanced codewords from the codeword list, as described in the following procedure:

```
$r = 0$, $s = 0$
DO $i = 1$ to $n$
    IF $c_i = 1$,
        THEN $r = r + 1$
        ELSE $s = s + 1$
END
IF $r - s \neq$ {one of predetermined set of values},
    THEN take this codeword off the codeword list
```

Note that the above set of values may be zero for codewords of even length, or (1, −1) for codewords of odd length.

In the above example with $l=4$, $n=7$, $b=4$, and the possible differences between the high and low level is +1 or −1, the mapping is as follows:

Data Word	Codeword
0001	0100111
0010	0100011
0101	0110011
0110	0110001
1000	0111001
1010	0010111
1011	0010011
1101	0011011
1110	0011001
1111	0011101

0000	0001011
0011	0001101

Note that the following data words are not mapped to any codeword, and therefore cannot be used: 0100, 0111, 1001, 1100.

3.4.3. Limited run–length

Generating the mapping table for the conservative code with limited run–length of m, is done as described before, with an additional step which eliminates the codewords with run–length greater than m from the codeword list, as described in the following procedure:

```
$r = 1$
$v = 0$
DO $i = 1$ to $n-1$
    IF $c_i = c_{i+1}$,
        THEN $r = r+1$
        ELSE $r = 1$
    IF $r > m$,
        THEN $v = 1$
END
    IF $v = 1$,
        THEN take this codeword off the codeword list
```

In the above example with $l=4$, $n=7$, $b=4$, and $m = 3$, the mapping is as follows:

Data Word	Codeword
0001	0100111
0010	0100011
0011	0100001
0100	0110111
0101	0110011
0110	0110001
0111	0111011
1000	0111001
1010	0010111
1011	0010011
1100	0010001
1101	0011011
1110	0011001
1111	0011101
0000	0001011
1001	0001001
Unused	0001101
Unused	0000101

3.4.4. Limited run–length with balancing

Generating the mapping table for the balanced conservative code with limited run–length of m is done as described before, with an additional step which eliminates the codewords with run–length greater than m, and unbalanced codewords from the conservative codewords list, as described in the following procedure:

```
$r=0$, $s=0$
DO $i=1$ to $n$
    IF $c_i = 1$,
        THEN $r = r+1$
        ELSE $s = s+1$
END
IF $r-s \neq$ {one of predetermine set of values},
    THEN take this codeword off the codeword list
```

```
$r = 1$

$v = 0$

DO $i$=1 to $n-1$

    IF $c_i = c_{i+1}$,

        THEN $r = r+1$

        ELSE $r = 1$

    IF $r > m$,

        THEN $v = 1$

END

    IF $v = 1$,

        THEN take this codeword off the codeword list
```

In the above example with l=4, n=7, b=4, and $m = 3$, the mapping is as follows:

Data Word	Codeword
0001	0100111
0010	0100011
0101	0110011
0110	0110001
1000	0111001
1010	0010111
1011	0010011
1101	0011011
1110	0011001
1111	0011101
0000	0001011
0011	0001101

Note that the following data words are not mapped to any codeword, and therefore cannot be used: 0100, 0111, 1001, 1100.

CHAPTER 4.

AN INTERFACE TO HIGH–SPEED MULTIPLE–ACCESS CHANNELS

4.1. General

Presented in this chapter the design of a serial electronic interface to a multiple–access fiber–optic network. Its goal is to maximize the interface bandwidth (greater than 1 gigabit/sec with GaAs technology), which is usually the bottleneck of an optical communication system ([OfFa87a]).

The design has five major objectives: (1) To maximize the transmission rate by reducing the critical timing path to one flipflop and one gate. (2) To minimize the number of flipflops which are directly driven by the transmission clock, so as to decrease clock skew. (3) To decode and perform the serial–to–parallel conversion of very small packets without the training period associated with a phase–locked loop. (4) To minimize the ratio between the transmission clock frequency and the baud rate.

These objectives are achieved by two techniques: (1) Hierarchical interface design – the interface design has four levels which successively reduce the data path width from the host bus to the optical link. (2) By using the 8B/12B conservative encoding scheme, as described in Chapter 3, characterized by a fixed number of transitions in every codeword. These transitions are used directly for decoding and for serial–to–parallel conversion, without having to recover the clock by means of a phased–locked loop.

Three general assumptions guide this design: (1) Hardware is not a scarce resource; it can be used to reduce the complexity of the system's operation and to improve its performance. For example, data encoding and CRC generation can be done on blocks of bits, not serially. (2) The medium bandwidth is higher than the digital electronic interface bandwidth. The parallel–to–serial conversion (a shift register with parallel load), and the decoder with serial–to–parallel conversion are the bottlenecks of the digital electronic interface. (3) The lines which connect the discrete devices are transmission lines with very accurate characteristic impedance, and these can be used as highly accurate and stable *delay lines.* The delays for this design, which are less than one nanosecond, can be realized by less than 20 cm of transmission line on a printed circuit board.

4.2. Comparison of the Conservative Code with Other Codes

The interface is designed for using the conservative code. In this section this code is compared with other codes. The major criterion is maximizing the bandwidth of the digital electronic interface, which is more critical than the bandwidth of the optical channel. Therefore, the objective is to have the DBBRR (digital bandwidth to baud rate ratio) as low as possible, i.e., to have maximum baud rate in a given technology characterized by typical and worst–case gate delay. Three criteria are used to evaluate different codes: (1) Resolution – the required sensitivity of the receiver for correct decoding, expressed in terms of a bit–cell. The resolution is the reciprocal value of the DBBRR. (2) DC level – the difference between the time that the transmitted signal is high and the time that the signal is low; it is desirable that this difference be as close as possible to zero. (3) Decoding without a PLL. One of the design objectives is to be able to decode an incoming packet without the training period required by a PLL, which can be substantial at high transmission rates.

The following is a review of some of the relevant self–clocking codes [Seve80], [Sore84]:

(1) Manchester Encoding – is a level type code in which a one has a high level at the beginning of the bit–cell with high–to–low transition at midcell, while a zero starts at a low level with low–to–high transition at midcell. Note that this code is not invertible.

(2) Biphase–Mark Encoding – is an edge–type code. Each bit–cell begins with an edge; for a one an additional edge occurs at midcell, while for a zero there is no additional edge. The code is invertible, since there is no rule for the polarities of the edges.

(3) Miller Encoding – is also called "delay modulation." This is an edge–type code; each one in the serial data is encoded as a mid–cell transition, while zero either has no transition (following a one) or is encoded with an edge at the beginning of the bit–cell. As a result, the encoded serial data have edges occurring at intervals of 1.0, 1.5 or 2.0 bit times.

(4) The 4B/5B encoding schemes – This code changes 4 bits of data into 5 bits of data. The motivation for this encoding scheme is to ensure enough transitions but not a fixed number. It is possible to choose a subset of 24 words such that no more than three consecutive zeros occur. If the bits are transmitted as non–return to zero inverted (NRZI), then there is no signal transition separation of more than three–bit cell periods. This scheme is almost balanced [JoIy84] and

has an efficiency of 80%, or bandwidth increase of only 25%.

The following table summarizes some relevant properties of the different encoding schemes:

TABLE 4.1. CODING SCHEMES COMPARISON

Scheme	DC	Efficiency	Decoding without PLL
Manchester/Biphase	none	50%	possible
Miller	almost none	50%	difficult
1–2 PWM (average)	almost none	67%	very simple
1–2 PWM (worst)	almost none	50%	very simple
NRZ	substantial	100%	impossible
4B/5B (with NRZI)	almost none	80%	very difficult
Conservative (8B/12B)	almost none	67%	very simple
Conservative (16B/20B)	almost none	80%	simple

The 8B/12B conservative code has three basic advantages: (i) the efficiency is better than 67%, (ii) the dc level is very low with a very high probability, and (iii) the decoding is very simple without a PLL.

4.3. Bandwidth Matching

In order to match the serial link and host bandwidths, the interface is hierarchically designed with four stages for managing the data path from the host bus to the optical link. At each stage the data path is controlled by a finite state machine. The requirement for matching is that between adjacent stages the stage closer to the host will be fast enough to serve the stage closer to the optical link.

Figure 4.1 describes the interface stages in terms of two major characteristics: (i) the technology required for realizing the stage, and (ii) the maximum time period (or interval) allowed for the transition from state to state. The following list specifies the main operation at each stage:

(1) Shift register level – single–bit interval, the time for changing the state of the shift register from shift to parallel load.

(2) Encoding/decoding – 8–bit interval, for performing these functions in parallel on data bytes.

(3) Transmitter and receiver finite state controllers – 64–bit interval, the time for writing or reading a word to/from the transmitter or the receiver.

(4) Buffer management, routing and access control unit – packet length interval for performing these functions in real time.

4.4. Functional Description

This section outlines the basic properties of the interface, as shown in Figure 4.2, and describes the principles of its operation. It is assumed that there is a set of buffers (or FIFOs) which can each store one packet of data. The buffers are dual–ported to the host bus and to the interface (receiver or transmitter). Access control, routing control, and buffer management are not included in the following description.

4.4.1. Principle of operation

The communication system has three major parts, as shown in Figure 4.3. The central part is the communication channel, which carries serial data. The channel transfers the serial bit stream from the encoder to the decoder.

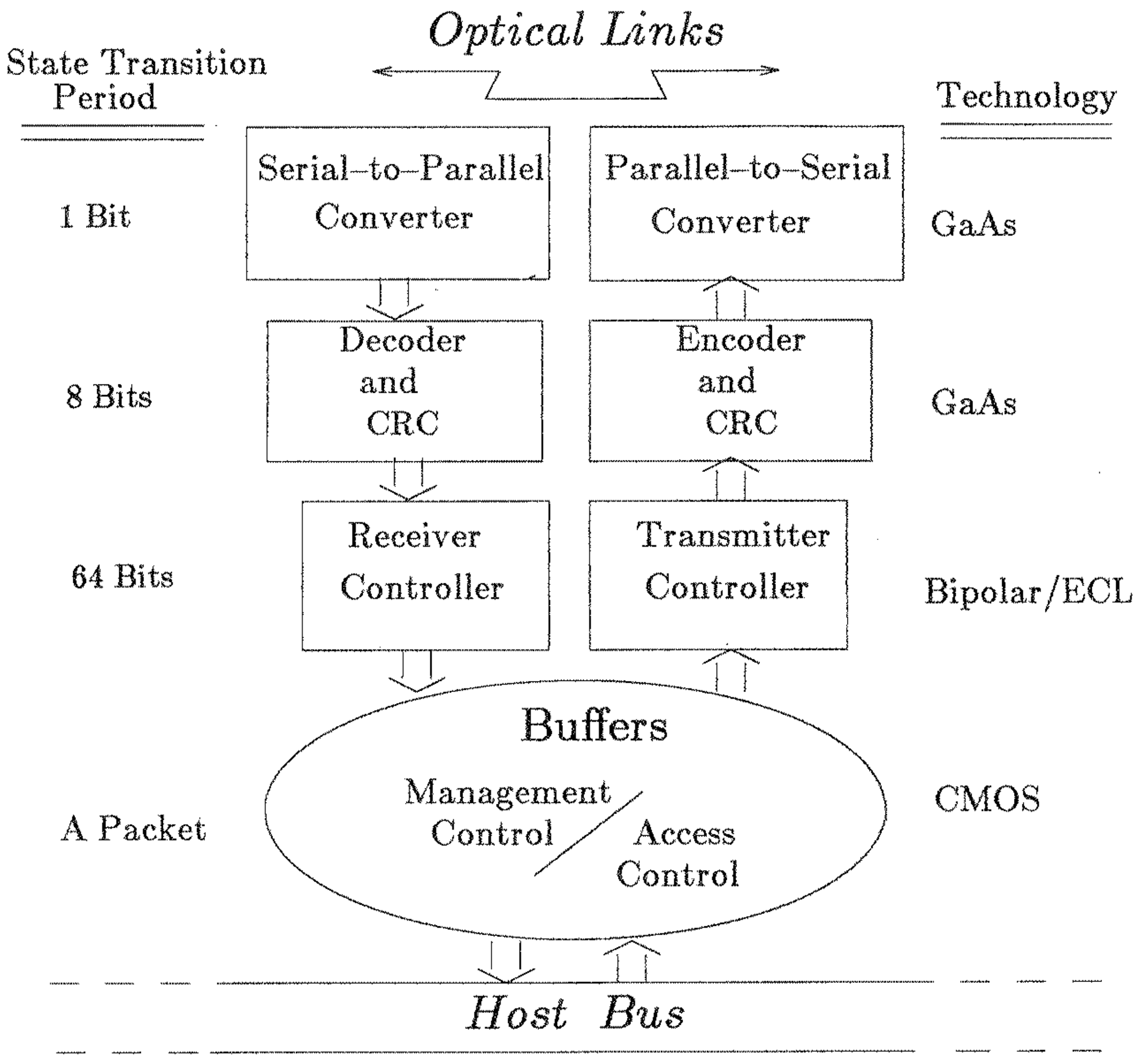


Figure 4.1: Bandwidth Matching

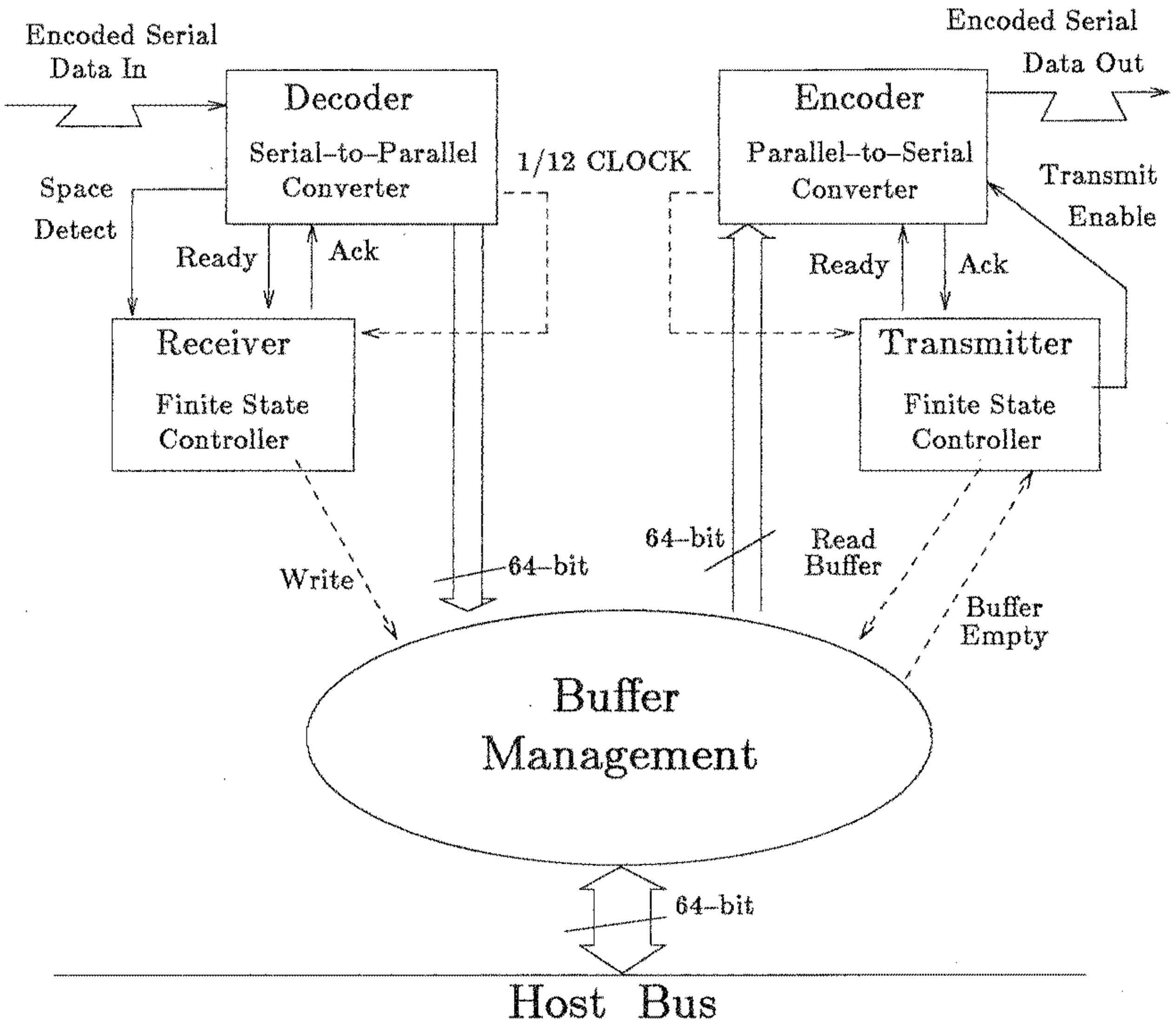


Figure 4.2: Functional Description of the Interface

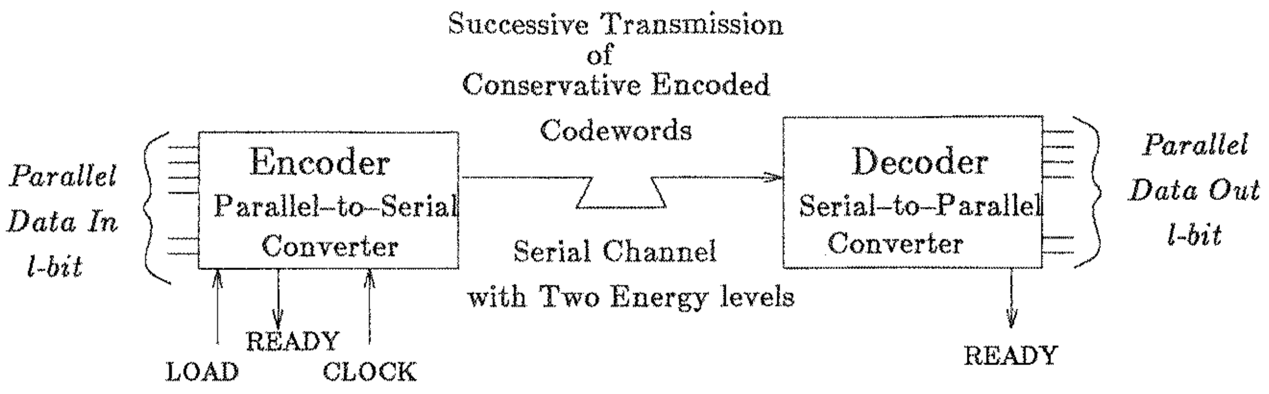

Successive Transmission of Conservative Encoded Codewords
Parallel Data In l-bit
Encoder
Parallel–to–Serial Converter
Serial Channel with Two Energy levels
Decoder
Serial–to–Parallel Converter
Parallel Data Out l-bit
LOAD
READY
CLOCK
READY

Figure 4.3: Communication Block Diagram

Note that the decoder and serial–to–parallel converter operate without explicitly recovering the receiving clock.

4.4.1.1. Clocking

A basic question is whether to use one or both edges of the clock signal in order to get better system performance. In general, clock generator modules have the following properties: (1) a clock stability and accuracy better than $\pm 0.1\%$, and (2) a clock symmetry accuracy of $\pm 5\%$. In high speed serial communication, in order to improve the reliability of the decoding and parallel–to–serial conversion, it is important that the output signal be as accurate as possible. In this design *only one of the two clock edges is used,* so that the output timing will not depend on clock symmetry. Therefore, the digital interface bandwidth is greater than or equal to the baud rate.

4.4.1.2. The packet header

The data to be transferred are organized into packets, which are continuous serial streams of bits. The packet header consists of one word of n bits; the first three quarters of these have some pattern of zeros and ones, and the last quarter of the header is a space, as shown in Figure 4.4. The receiver searches for this space and starts recording data only after it is detected. This technique reduces the effect of random noise, which might occur while the optical receiver is idle.

4.4.1.3. Full–duplex interface

The transmitter and receiver are implemented as two independent submodules, i.e., the network interface is *full-duplex*. This makes the design of the interface simple, reliable, and testable. An important consequence of this, for topologies like star, is that a receiver may check its transmitter output, using either an error–detecting code or direct packet–to–packet comparison. A full–duplex interface is necessary for collision detection in random access protocols, such as CSMA/CD.

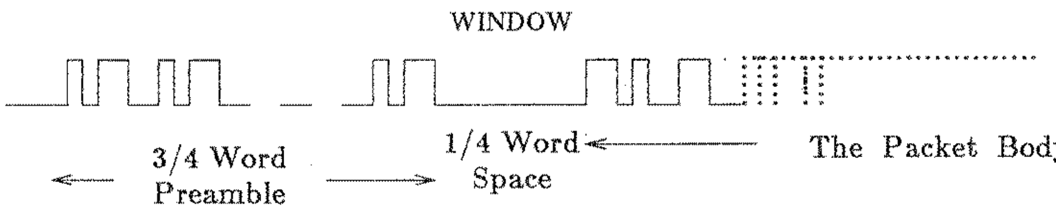


Figure 4.4: The Packet Head Pattern

4.4.2. The transmitter

The functional description of the transmitter is shown in Figure 4.5. The input register is reloaded, and the next word is encoded while the shift register transmits the current word.

The transmitter's data path is controlled by two finite state machines which perform the following functions: (1) Generate the packet header. (2) Encode the data in parallel on blocks of 8 bits by a combinational network. (3) Parallel–to–serial conversion. Load Next Word and Ready are the handshake signals between the transmitter finite state controller and the encoder.

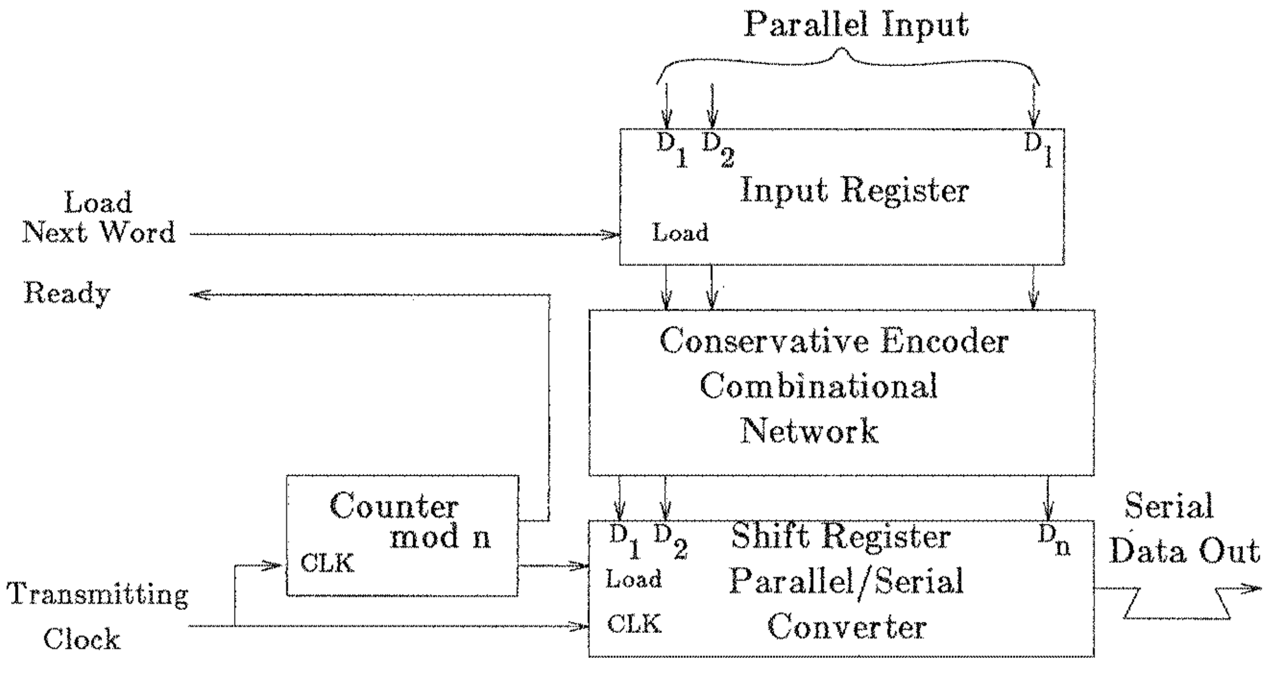


Figure 4.5: Functional Description of the Transmitter

4.4.3. The receiver

The functional description of the receiver is as shown in Figure 4.6. The receiver's data path is likewise controlled by two finite state machines, which perform the following functions: (1) Decode and serial–to–parallel conversion. (2) Check the correctness of the incoming packet. (3) Packet destination recognition; if the packet is not addressed to this node, the receiver discards it.

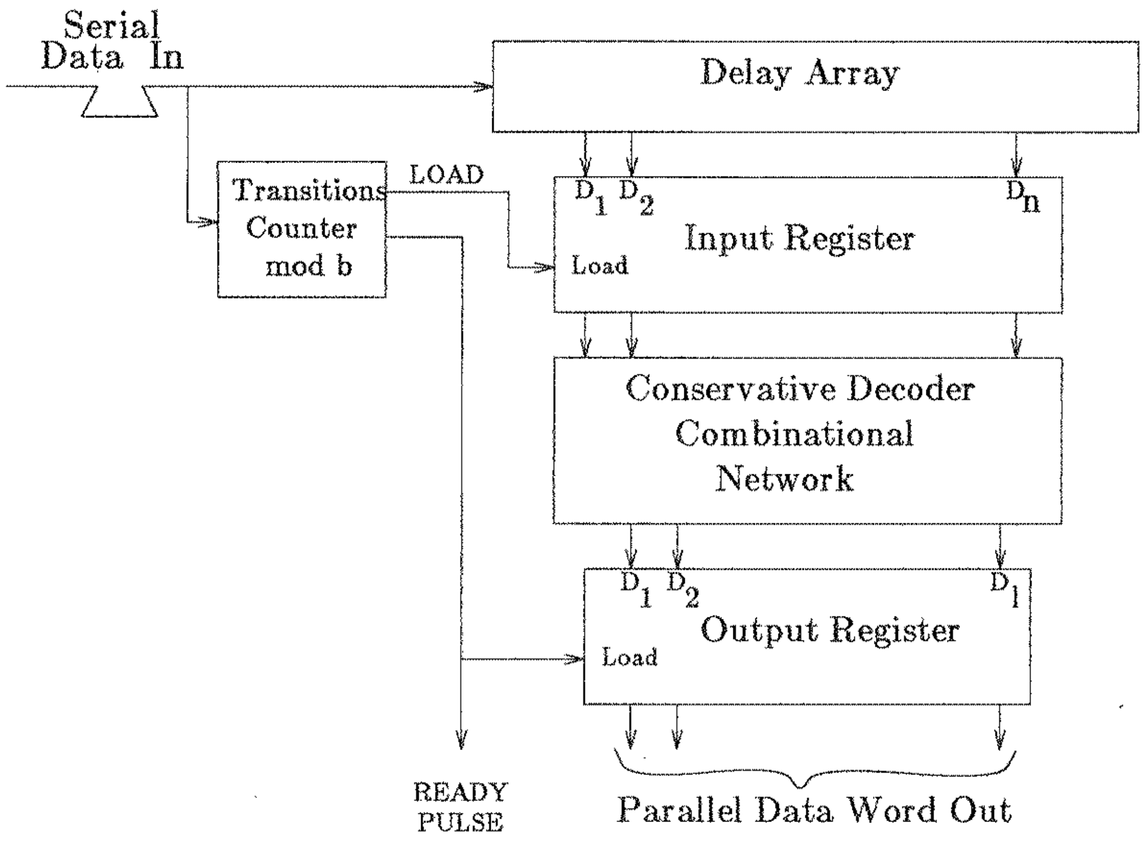


Figure 4.6: Functional Description of the Receiver

4.5. Encoder and Parallel–to–Serial Converter

Described in this section the details of the link level interface for either the 1–2 PWM or 8B/12B conservative encoding schemes. It could be implemented with GaAs technology and use a 1500 MHz clock for a transmission rate of 1 gigabit/sec.

In order to obtain maximum speed, the critical delay path is minimized to a shift register with a parallel load (see Figure 4.7), i.e., the *critical delay* is a linear summation (worst case analysis) of three delays: flipflop set–up time, t_s, flipflop propagation delay, t_d, and NOR–gate propagation delay, t_n. The first two delays are necessary for the shift register, and the third delay is required for the parallel load. The maximum frequency at which the interface can operate is therefore

$$BW_{\max}=\frac{1}{t_s+t_d+t_n}$$

Note that loading via a single gate with a wired–OR output is possible because the shift register contains only zeros by the time of the next load.

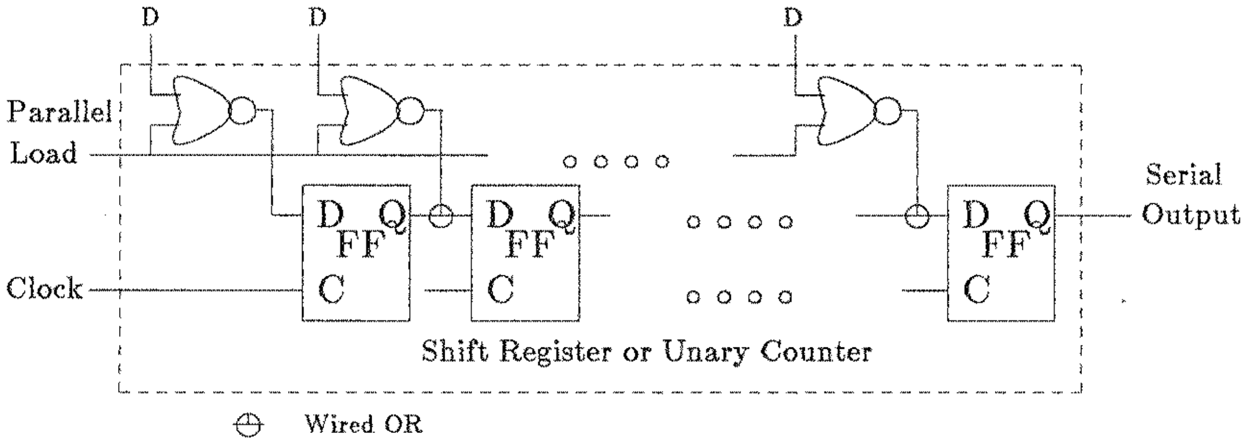


Figure 4.7: The Shift Register Design

Figure 4.8 (p. 57) is a block diagram of the transmitter's link interface. The **Shift Register Array** is part of the bandwidth matching mechanism. It is parallel–loaded with a 64–bit word from the **Data Bus.** The 8 least significant (rightmost) bits of the **Shift Register Array** are input to two **Combinational Networks.** The **Shift Register Array** is clocked by the /LOAD–BYTE signal. The **Word Load Unary Counter** counts eight /LOAD–BYTE pulses, on the 8^{th} of which LOAD–WORD signal is asserted. As a result, on the following /LOAD–BYTE pulse, the array is loaded with a new word from the **Data Bus.**

One of the **Combinational Networks** is the encoder, which receives 8 bits from the shift register array, and transforms them into a 12–bit codeword. If the START–ENABLE signal is deasserted, the output of this network is always zero. The other combinational network is the **Programmable Load Timer**, which receives the same 8 bits from the array and transforms them into an output word with a single 1, whose position determines the time of the next byte load. The output of this combinational network is loaded into the **Byte Load Unary Counter.**

In order to achieve maximum performance all counters are **unary**, i.e., shift registers with parallel–load (as shown in Figure 4.7). There are three unary counters in this interface: (i) **Byte Load** – for determining when to load the next byte into the **Parallel–to–Serial Converter.** (ii) **Word Load** – for determining when to load the next word into the **Shift Register Array.** (iii) **Space** – which is part of the **Start and Space Controller**, and ensures that the last two bytes of the preamble are transmitted as a continuous low level by deasserting the START–ENABLE signal.

The **Start and Space Controller** synchronizes the transmitter interface with the transmitter finite state controller. The START–ENABLE signal is asserted after the rising edge of the /LOAD–BYTE signal if both LOAD–WORD and TRANSMIT–ENABLE are asserted. The READY signal is asserted (by the finite state controller) whenever a new word is ready on the input data bus. The ACK signal is asserted after the rising edge of /LOAD–BYTE if LOAD–WORD is asserted (and the new word is loaded into the **Shift Register Array**). After the assertion of the ACK signal the READY signal is deasserted, and then ACK is also deasserted, to complete the handshake.

4.5.1. The operation of the transmitter link interface

The operation of the link interface has four basic phases: idle, start, running and termination, through which the **Shift Register Array, Parallel–to–Serial Converter, Byte Load Unary–Counter**, and **Word Load Unary–Counter** operate **continuously**. Only the **Start and Space Controller** changes its state during these four phases.

(1) *Idle phase* - TRANSMIT–ENABLE and START–ENABLE are deasserted. As a result, the **Serial–to–Parallel Converter** is loaded with zeros and the serial output is continuously low. The **Byte Load Unary–Counter** is loaded with 1 at the 12th position from the right, so that every 12 clock cycles (1500MHz) there is a /LOAD–BYTE pulse for one clock period. Every 8 /LOAD–BYTE pulses LOAD–WORD is asserted, and the **Shift Register Array** is loaded with 64–bit word from the **Data Bus**.

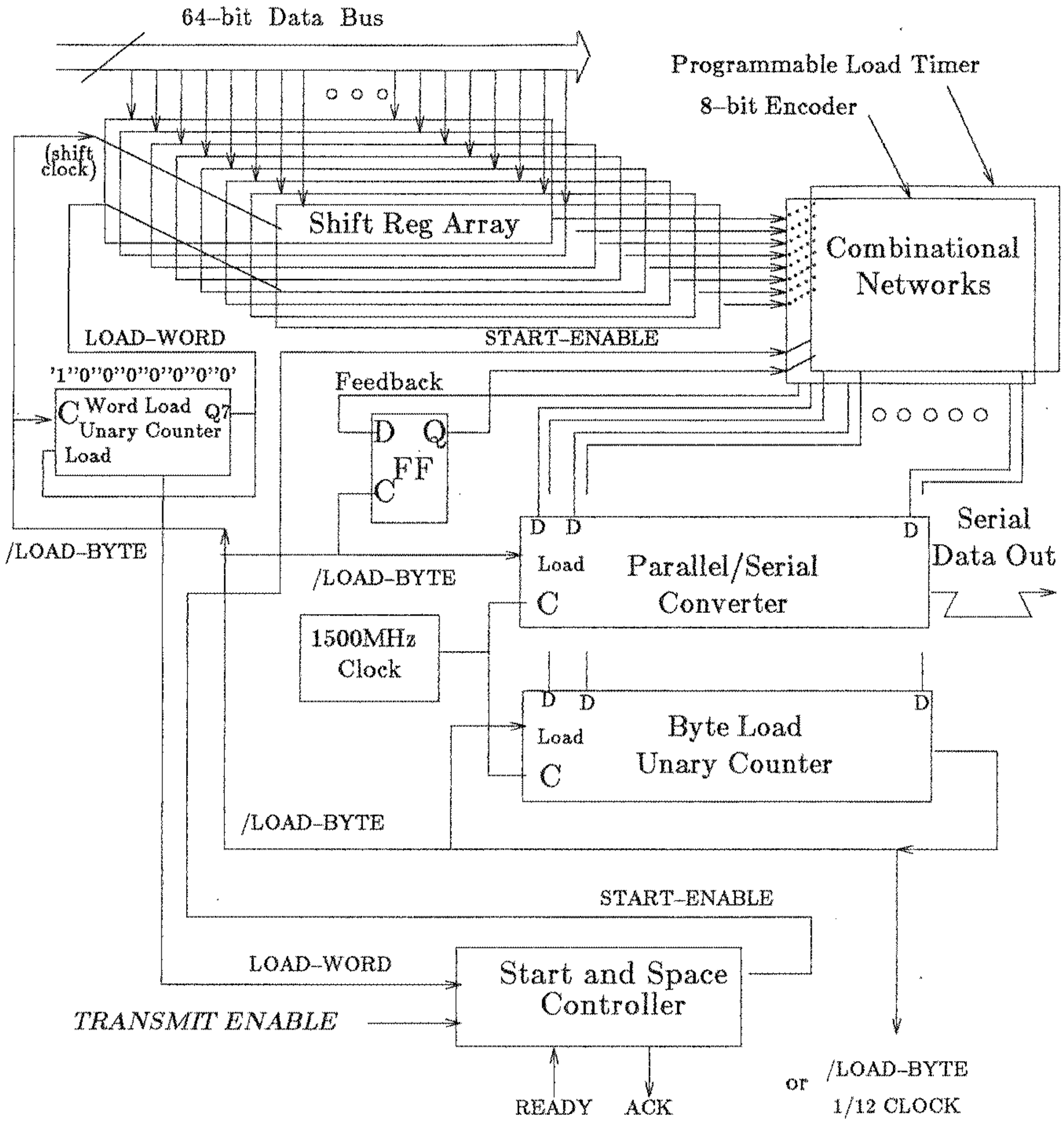


Figure 4.8: The Transmitter Link Level Interface

(2) *Start phase* - proceeds as follows: (1) TRANSMIT–ENABLE is asserted by the transmitter finite state controller. (2) START–ENABLE is asserted after the assertion of LOAD–WORD and /LOAD–BYTE. (3) Six bytes of the preamble are transmitted. (4) START–ENABLE is deasserted and two bytes of space are transmitted. (5) START–ENABLE is asserted and the first word of data is transmitted.

(3) *Running Phase* - data is transmitted continuously. (1) The ACK signal is asserted after the assertion of LOAD–WORD; the next word is loaded into the **Shift Register Array**. (2) The transmitter finite state controller deasserts READY. (3) The **Start and Space Controller** deasserts the ACK signal. (4) The transmitter finite state controller asserts the next data word onto the **Data Bus**, upon which READY is asserted. (5) Then the cycle repeats.

(4) *Termination phase* - after the last word is loaded, ACK causes the transmitter finite state controller to deassert both TRANSMIT–ENABLE and READY, and then ACK is deasserted. START–ENABLE is deasserted after the assertion of LOAD–WORD and the rising edge of the /LOAD–BYTE pulse.

4.6. Decoder and Serial–to–Parallel Converter

The basic principle of the interface operation, as previously explained, is to use the transition information of the incoming serial data both for decoding and for serial–to–parallel conversion. The receiver design for 8B/12B conservative encoding scheme is presented, and then some of the actual engineering issues are discussed.

4.8.1. The 8B/12B conservative decoder

A block diagram of the 8B/12B decoder for 1 gigabit/second is shown in Figure 4.9. The incoming serial data (Din) has exactly **three falling edges** for every 8 bits of data (or 12 bits of codeword). The third falling edge is the delimiter of the 12–bit codeword. Din is propagated via a transmission line (TL), which has 12 evenly spaced stubs as inputs to 12 flipflops. The Din signal also clocks the **Word Transition Unary–Counter.** This counter is designed for maximum bandwidth as described in Figure 4.7. The unary counter uses the falling edges of Din, and upon the third falling edge the /LOAD–BYTE signal is asserted, and then 12 bits are sampled from the TL into the 12 input flipflops.

The outputs of the 12 flipflops are decoded back to the original 8–bit data by the **Decoder Combinational Network**, and then fed into an array of 8 shift registers. There are two arrays of shift registers for double buffering, each containing a word of 64 bits. The signals CLK WORD1 and CLK WORD2 are used to clock the shift registers. Only one clock signal is active at a time, and they switch roles after 24 falling edges or after 8 rising edges of the /LOAD–BYTE signal (after 8 bytes of data are recorded). The word in the array which is not being clocked can be read by the receiver finite state controller into the input buffer. ACK and READY are handshake signals with the receiver finite state controller, and are used for synchronizing the transfers of words into the input buffer.

The **Space Detect** unit searches for the space between the preamble and the data. It uses a shift register with a 1500MHz clock for recording the incoming serial data into the shift register. A simple combinational network, which uses the shift

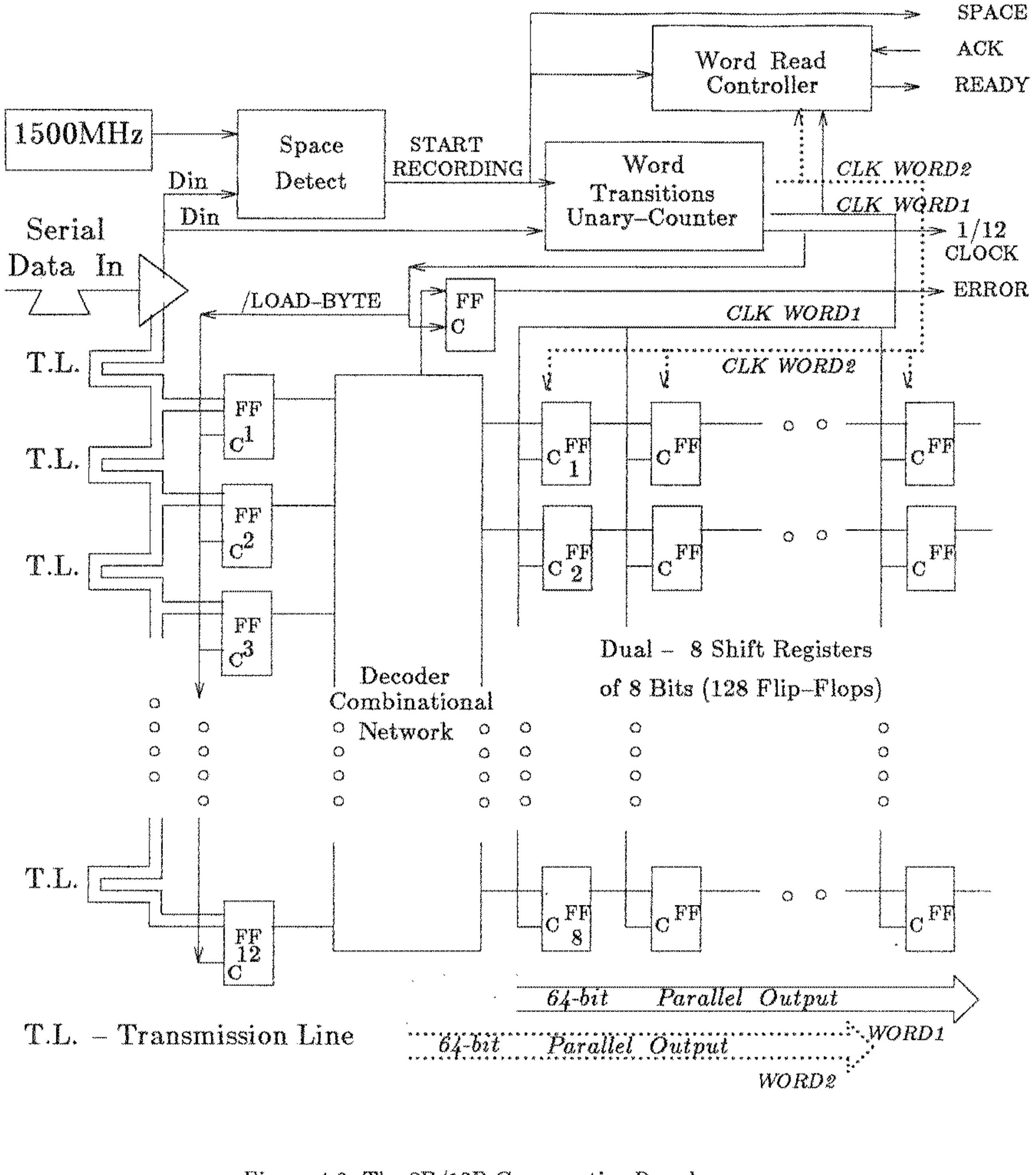


Figure 4.9: The 8B/12B Conservative Decoder

register outputs, is used to detect the space between the preamble and the body of the packet. This unit sends the signals START RECORDING or SPACE to the receiver finite state controller whenever the space is detected. This signal is also used for resetting the **Word Transitions Unary–Counter**.

Error detection is possible, since the codeword space is larger than the data word space (462 words vs. 256 words). If a codeword cannot be mapped to one of the data words, the ERROR signal is asserted. If a collision has occurred, the high levels of the Din signal are extended (transition information is destroyed), and with a high probability one of the succeeding input codewords is illegal.

The 1/12 CLOCK from the **Word Transitions Unary–Counter** is the clock signal to the receiver finite state controller, so that these two units are synchronized. While the input line is idle, the 1/12 CLOCK is derived from the local 1500MHz clock.

The 12 input flipflops can be realized on one integrated circuit, so they can be clocked simultaneously by the /LOAD–BYTE signal. The transmission line is realized on a printed circuit board (each of its segments is about 15–20 cm.).

4.6.2. Discussion

There are some engineering issues that affect the design, performance, and reliability of this interface. First, **group velocity dispersion** in the fiber causes pulse spreading $\Delta\tau$ proportional to the spectral width $\Delta\omega$ and the distance L:

$$\Delta\tau \approx \frac{2L}{v_g^2}\left|\frac{dv_g}{d\omega}\right|\Delta\omega$$

v_g is the group velocity, and $\Delta\tau$ determines the maximum bandwidth of the fiber. The pulse spreading distorts the serial signal shape, and makes the decoding and the serial–to–parallel conversion more difficult.

Another issue is the temperature stability of the TL, which determines the stability of the delays between the input flipflops. The PC board is thermally more stable than active devices, since it has a significant thermal inertia. The size and passive nature of the TL would suggest that it is much easier to stabilize at a constant temperature than either an active delay line or RC circuits.

It is important to note that the 1500MHz clock is not used for decoding or parallel–to–serial conversion. It is used only for control and error checking. The decoding and converting circuitry is completely asynchronous and can operate within the tolerances of the delay lines.

4.7. Conclusions

The previous discussion has shown the feasibility of constructing an interface that meets all its design objectives. The critical timing section of the transmitter is small enough to be implemented on a single GaAs gate–array, and the transmitting clock needs to drive only this chip. Similarly, the critical timing section of the decoder (the unary counter, input flipflops, and the shift/hold control) can be integrated on another single GaAs chip.

The use of unary counters (shift registers) is essential for achieving a maximum transmission rate in a given technology. They should be relatively small, in order to avoid clock skew delay, and the hierarchical design methodology makes this possible.

The new conservative encoding/decoding methodology is flexible, stable and simple enough to be used in a multiple–access medium. The decoder uses delays realized by transmission lines. This realization is simpler as the delay becomes shorter, and therefore favors higher baud rates (as long as the pulse dispersion is not significant).

CHAPTER 5.

THE DESIGN AND OPERATION OF A SINGLE NET

Described in this chapter the design and operational principles of a single net. The net is a multiple–access medium constructed of a centralized, passive, optical star and single mode fiber optic links (see Section 2.3). The communication over the net merges to one point in space, and then broadcasts back to all the net's nodes. The transmission is divided into fixed length–time slots, with one slot counter at each node, to keep track of time using a slot as its basic time step.

The net's operational principles and timing are derived from the centralized star topology, which enables simple synchronization of the net's operation, and also the conservative coding scheme, which enables the periodic exchange of state information, via very short control messages.

5.1. The Net's Operational Principles and Timing

5.1.1. Net timing

The optical star is constructed in a very small area – a few square inches. Thus, this center is used as the time reference for the net's nodes. Each time slot has a fixed time duration T_s, which is measured in bit periods, i.e., the maximum number of bits which can be transmitted within one time slot at a given baud rate. The one–way delay of a node i from the star center is Δ_i. Thus, the n nodes of each net may be regarded as lying on the circumference of an imaginary circle of radius R, such that

$T_R \geq \max\{\Delta_i$, such that $n \geq i \geq 1\}$, as shown in Figure 5.1. T_R is an upper bound of the node–to–center delay. This uniform arrangement is used for the net synchronization mechanism. Because of this circular symmetry the nodes are **indistinguishable** with respect to the center point, and the symmetry is later used for modeling a partial hypergraph as a centralized switch.

The slots are grouped into frames of duration T_f, with f slots per frame and $T_f = fT_s$, as shown in Figure 5.2. For synchronization purposes, the frame duration is equal to $2T_R$, i.e., T_f is greater than the delay from any node to any other. The slot duration is then $T_s = \frac{2}{f}T_R$. Although T_f depends on the physical size of the network, it should be noted that T_s can be chosen according to other system considera-

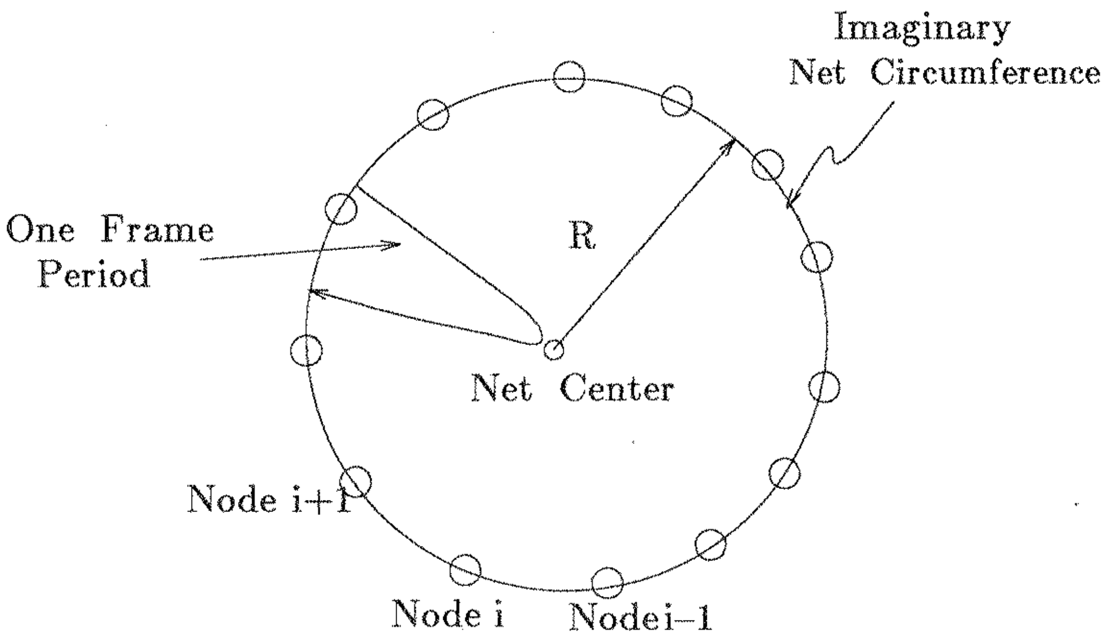


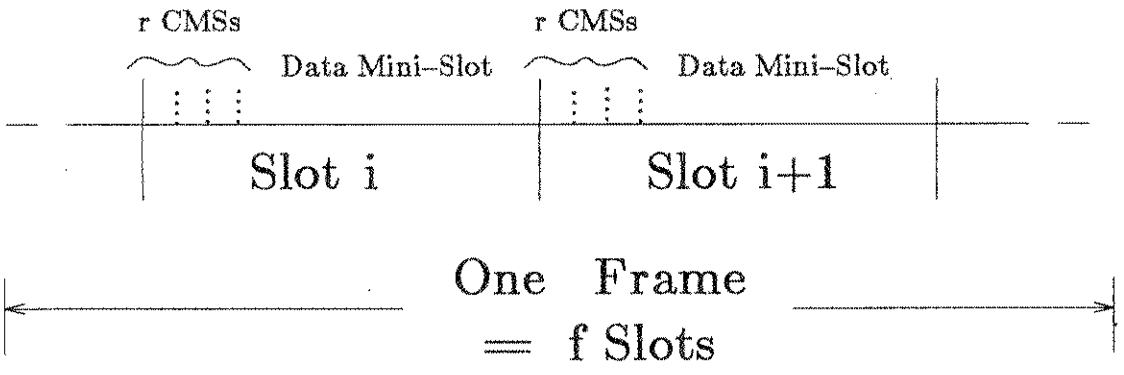
Figure 5.1: The Net Uniform Arrangement

tions. Once it is chosen, however, the net synchronization condition depends on f. $T_s = \frac{2}{f} T_R$.

5.1.2. Periodic exchange of state information

The basic operational principle of the net is the periodic exchange of state information. The major reason for having it is for making the integration of various distributed functions more uniform and efficient. In general, the control mechanism and the actual data communication and computation are independent. Therefore, it is reasonable to assume two basic requirements for the control mechanism: (i) periodic exchange, via (ii) short control messages.

The access control, synchronization mechanism, and the integration of other functions are based on the periodic exchange of timing and state information. To this end, each slot is divided into $r+1$ minislots, as shown in Figure 5.2. The first r are



CMS – Control Mini–Slot

Figure 5.2: The Frame and Slot Description

very short control minislots (CMSs), used for control. The last minislot, which occupies most of the slot space, is the data minislot (DMS), for transmitting one packet of data. The $r+1$ minislots of each slot are used by $r+1$ different nodes. The asymmetric partitioning of the slot between CMSs and one DMS makes it possible for one node to send "useful" data, and for r nodes to send very short messages, consisting of state information for the control of the communication and computation.

A message, sent during a CMS, is broadcast over the net from a **known origin** but without a **specific destination**. The access to the CMSs on each net is deterministic. For example, under a **uniform scheme** the set of n nodes is partitioned into r **disjoint subsets** of sizes either $\left\lceil \frac{n}{r} \right\rceil$ or $\left\lfloor \frac{n}{r} \right\rfloor$ nodes. The sum of the sizes of all these subsets is n, and each node belongs to exactly one of the subsets. The nodes in a subset use the corresponding CMS in a round–robin fashion.

For the following analysis it is important to note that each node can use its CMS: every $l = \left\lceil \frac{n}{r} \right\rceil$ time slots. As a result, each node can broadcast its *view* over the net every l time slots.

5.2. Hybrid Multiple Access Control Algorithm

The access control algorithm uses the load information which is exchanged periodically during the control minislots. The access control algorithm operates using two basic modes: **random** for light net load, and **deterministic** for high net load. The net dynamically switches its operational mode. Another hybrid scheme that operates in deterministic and random modes is proposed and analyzed in [GoWo85]. The random algorithm is a simple version of a nonpersistent (or p–persistent) access control scheme [see Tane81]. The deterministic algorithm is distributed, and is performed by each node's interface. The algorithm considers only the load information currently **known** by all the nodes on the net. As will be shown, the deterministic algorithm is adaptive and guarantees high utilization of the net's communication capacity.

An access control scheme is analyzed on the basis of how efficiently it utilizes its communication capacity. For slotted communication, two events should be minimized or avoided: (i) having empty slots while some ports have full buffers, and (ii) having collisions of two or more packets, which result in a wasted slot. The first event **is avoided** by the random access mechanism, and the second **is minimized** by the deterministic algorithm. Hence, in order to maximize the utilization, the deterministic algorithm will be used whenever there is a **known** load, and the random access mechanism will be used whenever there is **no known** load.

There is a reciprocal relationship between the network capacity utilization and the average network delay. For a given load pattern, *maximizing the utilization* is equivalent to *minimizing the average delay* (and vice versa).

5.2.1. The deterministic adaptive access control scheme

Let $\{node_1, node_2, \ldots\ldots, node_n\}$ be the set of n nodes of a single net. The set of nodes is divided into r subsets, each subset uses one of the r CMSs for updating its state information. Each node belongs to **at least one** of the subsets. In general, a node can be either **busy** – with one or more full buffers to send, or **empty** – with no packet to send. A node which is **empty** and then receives a new packet to send will change its state to **busy** only after its new load becomes **common knowledge**, i.e., after the new state information has been broadcast over the net, and has reached the imaginary circumference. It is assumed, in general, that there is an additional one-time slot delay in order to incorporate the new state information into the next state. The delay from the time a CMS message is sent until the state transition, which uses this information, is $f+1$ time slots. The state information sent during each CMS may include

(i) the node identification (its physical and logical addresses)

(ii) the number of packets which are ready to be sent from this port

(iii) the number of empty buffers at this port

(iv) the time stamp – the current local slot counter reading

(v) the average node's load during the past s time slots

(vi) global synchronization information

5.2.2. Priority of the control minislots

The way a net is partitioned into control minislots can implement a priority scheme. The priority is viewed here as the average delay of a node for changing its own state information, which depends on how often the node gets a CMS. Two basic partitioning methods can be used:

(i) different-sized subsets nodes with high priority would be in a smaller subset, and therefore, would receive the CMS more often.

(ii) nondisjoint subsets nodes with high priority would be in more than one CMS subset.

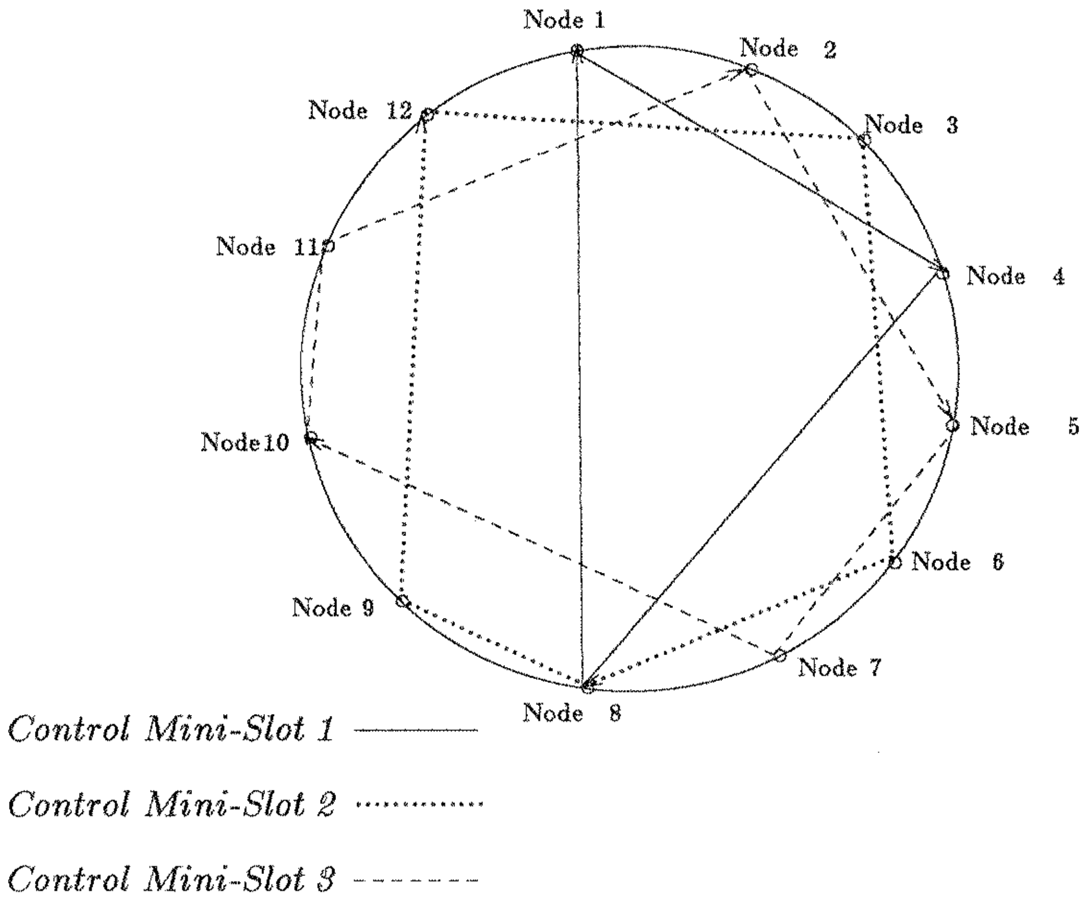


Figure 5.3: An Example for CMSs Partitioning

Figure 5.3 is a possible partitioning of 12 nodes of a net into three subsets. The three subsets are not of the same size, and a node can belong to more than one subset (e.g., node 8). In this example there three priority levels: level 1 – $\{node_8\}$, level 2 – $\{node_1,\ node_4\}$, and level 3 – $\{node_2,\ node_3,\ node_5,\ node_6,\ node_7,\ node_9,\ node_{10},\ node_{11},\ node_{12}\}$. Nodes which generate more traffic and act as "masters" are likely to have a higher priority in order to improve the system's response time, and hence its efficiency. In a real time application, a node on a critical timing path is likely to get a higher priority in order to improve its response time.

5.2.3. The scheduling during the data minislot

In the deterministic access control scheme, one of the nodes that is **known** to be in the busy state sends a packet during the data minislot (DMS). This access rotates among the other known busy nodes. Several different scheduling schemes can be implemented:

(i) round–robin scheduling

(ii) scheduling with a fixed priority among the busy nodes

(iii) scheduling as a function of the number of full packets of the busy nodes

(iv) scheduling as a function of the number of empty buffers of the busy nodes

(v) some combination of the above

The desired priority scheme should have two basic properties:

(1) Fairness – a packet from any node under worst–case scenario would be sent after finite number of time slots.

(2) Decreased overflow probability – the number of buffers at each node is finite, and the transfer rate from the interface to the host storage is slower than the net bandwidth. Note that these two requirements are in conflict; if the fairness is increased then the overflow probability is also increased.

For example, assume that the priority scheme gives the highest priority to the nodes with more than ten full buffers. As a result, the probability of overflow is likely to be low, but there is a possibility for starvation of nodes with only a few packets. On the other hand, the round–robin scheduling is very fair, but the overflow probability can be relatively high. As will be shown in Chapter 7, overflow can be prevented by using a special flag bit in the CMS control message.

In general, in the following analysis and examples a uniform round–robin scheduling is assumed during CMSs. Thus, each node updates its state information every $l = \left\lceil \frac{n}{r} \right\rceil$ time slots.

5.2.4. The random access scheme

The load is considered to be **light** whenever there are no **known busy** nodes, and then the net's nodes switch their operational mode, during the data minislot (DMS), to the random accessing mode. Note that the accessing mechanism during the CMSs remains unchanged.

In the random access scheme, all nodes which have a full buffer (not yet **commonly known**) will transmit it in the next DMS with the probability p. If more than one node transmits at the same time, a collision occurs. To resolve this conflict, the nodes involved will retry to transmit with the probability p' ($p' \leq 0.5$) after $f+1$ slots (after the collision becomes commonly known). The access control will continue in the random mode as long as there are no **known** nodes in the **busy** state.

5.3. The Analysis of a Single Net

The following terms will be used in the analysis:

(i) μ_{net} – the efficiency of the use of the communication capacity for data transfers, or the ratio of the duration of data minislots to the total duration of the slot. Note that under this definition it is assumed that the messages during the CMSs are not "useful" data transfers.

(ii) d_{state} – the delay of a node changing its state from **empty** to **busy**, using its CMS message. This time is measured from the time an **empty** receives a new packet until the node becomes **busy**.

(iii) d_{access} – the delay in accessing the net (the time interval from the moment the node changes its state from **empty** to **busy** until the time it gets the first access right), and

(iv) $d_{src-dst}$ – the total time delay of a packet from source to destination.

In most cases, only the mean time analysis will be performed. This is sufficient, since the operation of the net is based mainly on the deterministic access control algorithms and round–robin scheduling. The objective of this analysis is to get an indication of the expected performance of such a system.

In the discussion, the following **basic model** is assumed:

(i) the basic unit of time is a slot

(ii) n identical nodes on each net

(iii) f slots in every frame

(iv) r control minislots (CMSs) in every slot

(v) the set of n nodes are partitioned into r **disjoint subsets** of sizes either $\left\lceil \frac{n}{r} \right\rceil$ or $\left\lfloor \frac{n}{r} \right\rfloor$ nodes, such that the sum of the subset sizes is n

(vi) one data minislot (DMS) in every slot

(vii) only the CMSs are used for exchanging state information; the DMS is used only for data transfers

(viii) the basic scheduling scheme during the DMS is round–robin

(ix) one packet at a time is sent by each node, i.e., the node becomes **busy**, sends one packet, then becomes **empty**, and so on. This last assumption presents the net performance as poorer than in actuality, since each node could have several packets to send each time it becomes **busy**.

Definition – Heavy Load:–

the net is in the heavy load state during the period of time (in slots) in which the **known** number of full buffers is one or more, or when at least one node is in the **busy** state. Thus, when the load is heavy, the access control is **deterministic**.

Definition – Light Load:–

the net is in the light load state during the period of time in which the **known** number of full buffers is *zero,* or when all the net's nodes are in the **empty** state. Thus, when the load is light, the **random mode** is used.

5.4. Communication Efficiency

The net synchronous organization guarantees, in principle, that in each instance a bit of information can be transmitted into the net from one of the net's interfaces. However, the communication efficiency is reduced synchronization timing error (Δ_s), discussed separately in Chapter 6, and by the presence of control minislots, which constitute a communication and computation management overhead. Let T_{CMS} be the duration of the CMS, and T_{DMS} be the duration of the DMS, then the **maximum communication efficiency** is

$$\mu_{com_{max}} = \frac{T_{DMS}}{T_{DMS} + rT_{CMS}} 100\ (\%).$$

In this formula it is assumed that all the DMSs have one packet of data. For example, if r=4, CMS duration is a 256 bits, and the DMS duration is 4Kbytes, then the communication efficiency is 97%.

5.5. Delay Analysis

5.5.1. State change delay

The delay in changing the node's state information (e.g., changing its state from **empty** to **busy**), is the time it takes for **locally known** information to become **commonly known**. On the average, this time is

$$d_{state_{ave}} = \left\lceil \frac{1}{2}\frac{n}{r} \right\rceil + f + 1 \ slots.$$

The expression $\left\lceil \frac{1}{2}\frac{n}{r} \right\rceil$ is the average time until the node has its next turn to send a control message during its CMS. The $f + 1$ slots is the time it takes the control message to be propagated and decoded by all the destinations. The upper–bound (worst case) on this time is

$$d_{state_{max}} = \left\lceil \frac{n}{r} \right\rceil + f + 1 \ slots.$$

5.5.2. Access delay

The access delay is the time for a node, in the **busy** state, to send a packet into the net. If on the average, in the heavy load case, there are m nodes in the busy state, then under round–robin scheduling the average time for a packet to be transmitted into the net is

$$d_{access_{ave}} = \left\lceil \frac{m}{2} \right\rceil \textit{slots}.$$

The upper–bound on this delay is

$$d_{access_{max}} = n - 1 \textit{ slots}.$$

5.5.3. Source–destination delay

The total time to send a packet from a node which is in the empty state to its destination, is on the average

$$d_{src-dst_{ave}} = \left\lceil \frac{1}{2}\frac{n}{r} \right\rceil + \left\lceil \frac{m}{2} \right\rceil + 2(f+1) \textit{ slots}.$$

The $(f+1)$ is added twice: for the transmission delay of the control message, and for the transmission delay of the packet.

The upper–bound on the total delay is

$$d_{src-dst_{max}} = \left\lceil \frac{n}{r} \right\rceil + n - 1 + 2(f+1) \textit{ slots}.$$

5.5.4. Source–destination delay with reservation or look–ahead

In real–time applications, the time required for transferring a packet from its source to destination can be significantly reduced. If a slot counter is used, a specific time slot can be reserved, and then the delay will be only $f+1$ slots.

Without a slot counter, the node can look ahead by changing its state from **empty** to **busy**, and then the delay will be $\left\lceil \frac{m}{2} \right\rceil + (f+1)$ slots, so the time to change the state is saved.

5.5.5. The delay as a function of the bandwidth and l

The actual time delay over the net depends on three basic factors: the bandwidth, the ratio $l = \left\lceil \frac{n}{r} \right\rceil$, and the traffic model. The first two are determined by the optical architecture, and the third one is determined by the way the system is used. In this section the delay is computed as a function of the first two parameters.

Assume that

(i) the slot duration is 4Kbytes or 32Kbit periods,

(iii) the bandwidth – BW is expressed in kilohertz,

(iii) the R is 10,000 meters,

(iv) the T_R is 0.05 millisecond, and

(v) the number of slots in a frame $f = \left\lceil \frac{2(0.05)}{\frac{32{,}768}{BW}} \right\rceil = \left\lceil \frac{0.1BW}{32{,}768} \right\rceil$

Thus, the maximum delay for the propagation of state information is

$$d_{state_{\max}} = (l+f+1)\frac{32{,}768}{BW} = (l+1)\frac{32{,}768}{BW} + \left\lceil \frac{0.1BW}{32{,}768} \right\rceil \frac{32{,}768}{BW} \approx$$

$$\approx (l+1)\frac{32{,}768}{BW} + 0.1\ millisecond.$$

The diagram in Figure 5.4 describes the delay as a function of the BW. Three cases are shown l=12, l=17, and l=22.

It is clear from Figure 5.4 that the state delay propagation is not more than a millisecond in a high bandwidth. This time is much faster than the typical access time to a magnetic disk. Note that the delay which is due to the physical size of the network is relatively insignificant.

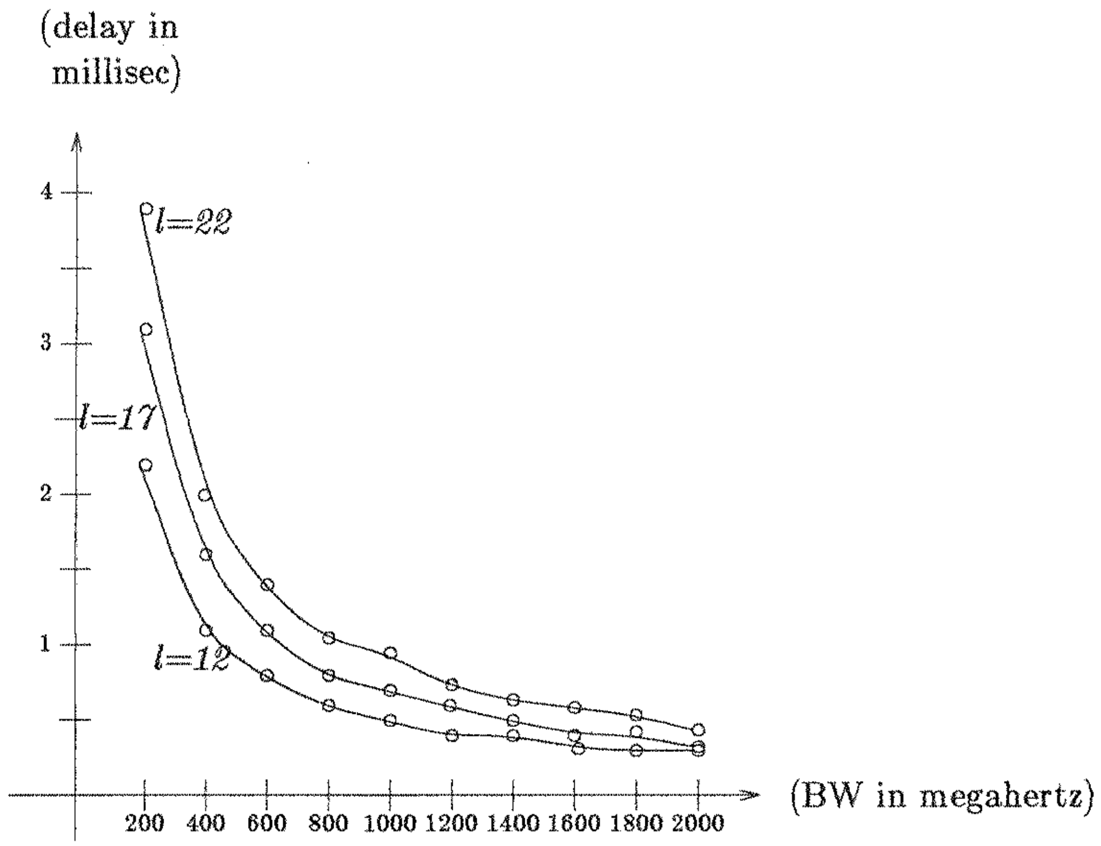


Figure 5.4: Maximum State Delay

5.6. Light Load Analysis

5.6.1. The model

Light load is analyzed on the basis of the efficiency of the communication over the net when the **known load is zero**. During this time the access control is random. It reverts to deterministic access, after new load information is received via the CMSs.

For the analysis the following model is assumed:

(i) the net has n identical nodes

(ii) the packet arrival at each of the n nodes follows Poisson statistics, with a mean arrival rate of λ per time slot (such that $n\lambda < 1$)

(iii) the Poisson process has the following characteristics:

- the numbers of arrivals in disjoint time intervals are **independent** random variables
- the number of arrivals is proportional to the duration of the observation time interval
- the superposition of Poisson processes is Poisson, with the mean arrival rate as the sum of the individual arrival rates
- the randomized thinning of a Poisson process is a Poisson process

(iv) $f = 1$ (frame of one slot)

(v) the set of n nodes is partitioned into r **disjoint subsets** of sizes either $\left\lceil \frac{n}{r} \right\rceil$ or $\left\lfloor \frac{n}{r} \right\rfloor$ nodes, with the sum of the subsets sizes equal to n

(vi) the result of the transmission (success, collision, or new **commonly known** load) is **known** by the end of each slot (thus, a node can retry to transmit in the subsequent time slot); also the control messages are decoded by the end of each time slot, and their information is incorporated into the next state transition

(vii) a node with a full buffer will try to transmit with the probability p in the first time slot of the random access mode, and with probability p' after a collision ($p' < 0.5$). Note that after a collision only the nodes involved will try to transmit.

(viii) state information is exchanged only via the r CMSs

(ix) in the random access mode, if a node has exactly one buffer full and can use its CMS, then it will transmit the data packet at random and declare its load using its CMS.

Let $l = \left\lceil \frac{n}{r} \right\rceil$, and divide the set of n nodes into r disjoint subsets $\{A_1, A_2, ..., A_r\}$. The nodes in each subset are arranged in a temporal order in which they last use their CMS

$A_i = \{a^i{}_{1T_s}, a^i{}_{2T_s}, ..., a^i{}_{lT_s}\}$; i.e., the $a^i{}_{lT_s}$ did not use its CMS for l time slots.

The probability that a node a_j has no packets after j time slots is

$$P_{j_{no-packet}} = e^{-\lambda j T_s},$$

and the probability that a node a_j has at least one packet after j time slots is

$$P_{j_{packet}} = 1 - e^{-\lambda j T_s}.$$

5.6.2. The probability for successful transmission

The probability for successful transmission during the **first** time slot after the **known load** becomes zero (all the nodes are empty), is computed in this section.

The probability that a node a_j will not transmit (idle) is

$$P_{j_{idle}} = 1 - pP_{j_{packet}},$$

this probability includes the cases (i) the node being empty, and (ii) the node being not empty but not transmitting $(1-p)$.

The probability that none of the nodes on the net will transmit (all the net's nodes are IDLE) is

$$P_{IDLE} = \left[\prod_{j=1}^{l} P_{j_{idle}} \right]^r$$

The probability that only the node a_j will transmit (node j ACTIVE) is

$$P_{j_{ACTIVE}} = pP_{j_{packet}} \frac{P_{IDLE}}{P_{j_{idle}}}$$

Finally, the probability for successful transmission in the first time slot after the net has changed its mode of operation to random access, is

$$P_{SUCCESS} = r\sum_{j=1}^{l} P_{j_{ACTIVE}}.$$

The probability for not having a collision is

$$P_{NO-COLLISION} = P_{SUCCESS} + P_{IDLE} = r\sum_{j=1}^{l} P_{j_{ACTIVE}} + \left[\prod_{j=1}^{l} P_{j_{idle}}\right]^{r}.$$

Thus, the probability for a collision is

$$P_{COLLISION} = 1 - P_{NO-COLLISION}.$$

5.6.3. The probability for a wasted slot

A slot is wasted when one or more of the nodes have packets to transmit, and there was no successful transmission, either because of a collision or because the net was idle and not empty, i.e., a slot is wasted unless there was a successful transmission or the net was empty.

The probability for an empty net during the first time slot of the random access mode is

$$P_{EMPTY} = \left[\prod_{j=1}^{l} P_{j_{no-packet}}\right]^{r}$$

Thus, the probability that the first slot will be wasted is

$$P_{WASTE1} = 1 - P_{EMPTY} - P_{SUCCESS} = 1 - \left[\prod_{j=1}^{l} P_{j_{no-packet}}\right]^{r} - r\sum_{j=1}^{l} P_{j_{ACTIVE}}$$

In the design of the net the objective is to minimize the probability of a wasted slot during the random access mode, and by this to maximize the utilization (during the deterministic mode there are no wasted slots). If λ, r, and l are some system parameters, then p should be selected by simulation such that the probability for a wasted slot is minimized.

5.6.4. The second time slot

Three cases are possible in the second slot of the random access mode.

Case 1 – the transmission in the first time slot was successful, and the load of the node decreases (since $n\lambda < 1$). Thus, the probability for a wasted second slot is less than in the first time slot ($P_{WASTE2} < P_{WASTE1}$).

Case 2 – the net was idle in the first time slot. The probability for a wasted slot remains the same ($P_{WASTE2} = P_{WASTE1}$).

Case 3 – there was a collision during the first slot. Since p'–persistent algorithm is used, only the nodes which were involved in the collision will try again in the second time slot with probability p'. If m nodes were involved in the collision, then the probability for a wasted slot is

$$P_{WASTE2} = 1 - P_{SUCCESS2} - P_{IDLE2} = 1 - m\,(p'(1-p')^{m-1}) - (1-p')^{m} .$$

5.7. Discussion

The criterion for switching the access control between deterministic mode and random modes is optimal, if the access control objective is to minimize the number of wasted slots. In fact, it is not hard to prove the theorem that operating under the random access mode **only** while there is **no known load** is optimal.

Proof: the theorem is proved by negation; there are two cases.

Case 1 – the net switches its operation to random while there are one or more **known** full buffers. Since some nodes might also have full buffers which are **not known**, the probability for collision is greater than zero. Thus, a slot might be wasted.

Case 2 – the net switches its operation to random mode one or more slots after the last **known** full buffer was sent. Clearly, in this case one or more slots will be wasted. Thus, the switching criterion is optimal.

Similar arguments are valid for switching back from the random mode to the deterministic mode. In the random mode the p–persistent protocol is not the optimal protocol. It is, however, reasonable to use, since the protocol is very simple to implement and the load is light.

The maximum efficiency of the hybrid access control algorithm is almost 100%, since under heavy load the access control is deterministic.

CHAPTER 6.

DISTRIBUTED GLOBAL EVENT SYNCHRONIZATION

6.1. Introduction

Initially the system is asynchronous, and each node has its own local transmission clock. The objective of the synchronization procedure is to generate a global synchronous system clock. The method of achieving global synchronization is by fixing the slot duration in all the system's nets. The synchronization procedure proceeds in the following steps: (i) determining two synchronization conditions, (ii) designing two synchronization algorithms for matching the phases of the time slots on orthogonal nets, and (iii) analyzing these algorithms ([OfFa87b]).

The result of maintaining global synchronization among all the system's nets is that this distributed system preserves a **total global ordering** of all the events in the system. This total ordering is achieved with a small synchronization overhead (less than 5%).

Global event synchronization is a fundamental operating principle used for communication management and for concurrency control. It is simpler and more efficient to implement these functions under a synchronous scheme rather than an asynchronous one. The network architecture is a hypergraph which makes the synchronization simpler, and more reliable and efficient than a point–to–point network.

Each node determines time by using a slot counter. The values of the slot counters are used for stamping all computation and communication events. As a result, at any later time, on any node, and for all pairs of events A and B, it is possible to determine whether A was before, after, or simultaneous with B.

Global system synchronization enables a **well–defined global state transition** at the end of each time slot, which simplifies the operation of distributed network control algorithms (e.g., routing, buffer management), and the concurrency control of parallel processing transactions, i.e., time stamps can be used for maintaining the serializability of parallel algorithms. For example, time stamps are used in optimistic concurrency control algorithms for distributed database systems (see [KuRo81]). This algorithm is simple and enables a high degree of parallelism, higher than for techniques using only locks.

Global synchronization of a regular hypergraph and partial hypergraph, is modeled and analyzed in this chapter. The objectives of the analysis are to find the necessary conditions for maintaining global synchronization, and to find the average and worst–case synchronization efficiency for a single net and for the whole network, i.e., the time overhead for maintaining synchronization.

The broadcast to all the net's nodes from one point in space has the inherent advantages of efficient distributed synchronization and fault tolerance (Chapter 8). It will be shown that the distributed synchronization technique leads to a small reduction in the number of bits that may be transmitted in a given time interval. The synchronization efficiency is defined as the ratio of this number to the maximum number of bits that could be transmitted in that interval over the shared medium.

Each time slot has a fixed time duration T_s. The one–way delay of a node i from the star center is Δ_i. Thus, the n nodes of each net may be regarded as lying on the circumference of an imaginary circle with radius R, such that $T_R \geq \max\{\Delta_i$, such that $n \geq i \geq 1\}$, as shown in Figure 5.1. The slots are grouped into frames of duration T_f, with f slots per frame and $T_f = fT_s$, as shown in Figure 5.2. For synchronization purposes, the frame duration is equal to $2T_R$.

The synchronization mechanism is based on the periodic exchange of timing and control (or state) information, which is also used for other network functions. Each slot is divided into *r+1 minislots.* The first r are very short control minislots (CMS) used for network control.

A message sent during a CMS is broadcast over the net from a **known origin** but without a **specific destination**. The use of the CMSs on each net is deterministic. The set of n nodes is partitioned into r **disjoint subsets** of sizes either $\left\lceil \frac{n}{r} \right\rceil$ or $\left\lfloor \frac{n}{r} \right\rfloor$ nodes. The sum of the sizes of all these subsets is exactly n, and each node belongs to exactly one of the subsets. The nodes in a subset use the corresponding CMS in a round–robin fashion.

In the following analysis it is important to note that each node can use its CMS every $l = \left\lceil \frac{n}{r} \right\rceil$ time slots. As a result, each node can broadcast its *timing view* every l time slots. The *timing view* will be discussed later, and consists of the time difference observed by a node over its two orthogonal nets. The *timing views* are sent in order to match the phases of all the nets in the network.

Other synchronization procedures which are found in literature are more general and abstract, and use explicit message-passing protocol, e.g., the Byzantine Agreement which is extensively discussed in literature. A general analysis of the complexity of network synchronization is found in [Awer85]. The following synchronization scheme is based on:

(i) specific optical architecture and its parameters, and

(ii) implicit and explicit message passing.

6.2. Distributed Synchronization

6.2.1. The synchronization hierarchy

Global distributed synchronization is achieved in a hierarchical manner, as shown in Figure 6.1. The lowest synchronization level is the net. There is a slot timer at each port, which is used to measure the duration of one time slot (T_s). Knowing its delay from the net center enables each port to determine when the next time slot should begin. The next synchronization level is done in a distributed manner by matching the timing differences between the two ports at each node.

Initialization of all the slot counters is centralized and done only at startup. The concurrency control level is the distributed operating system, which uses the underlying global synchronization. The lower levels are completely transparent to the concurrency level, and can be viewed as network functions. In general, the network functions use the CMSs, while the distributed operating system uses the DMSs.

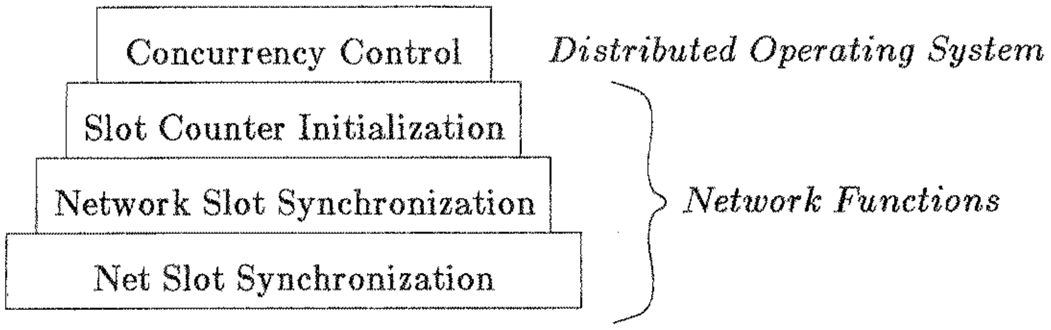

Concurrency Control
Slot Counter Initialization
Network Slot Synchronization
Net Slot Synchronization
Distributed Operating System
Network Functions

Figure 6.1: The Synchronization Hierarchy

6.2.2. Synchronization conditions for 2D hypergraph

In this section the necessary and sufficient synchronization conditions for ensuring a total temporal event ordering in the system are discussed. It will be shown how these conditions can be achieved and what the consequences are for the system's performance efficiency.

The following are the necessary and sufficient conditions for total event ordering:

SC1 – Synchronization Condition 1 (local condition):

At all times the reading of each slot counter (C_i) is not greater than the actual number of slots that have *passed* at each of the node's ports, and can be smaller by one, for durations less than half a time slot. As a result, the slots of the two ports can be **paired**, such that the time difference between the two paired slots, (one from each port), is less than half a time slot, i.e., the two streams of events (or slots) which each node "sees" via its two ports can be **uniquely paired**.

SC2 – Synchronization Condition 2 (global condition):

At all times the difference between any two slot counters C_i and C_j can be **zero** or **one**, i.e., at each time step there is an interval greater than half a time slot in which all the system's slot counters have the same value. Figure 6.2 presents a global view of the system's timing. During time interval d ($d < 0.5T_s$) all the slot counters are incremented.

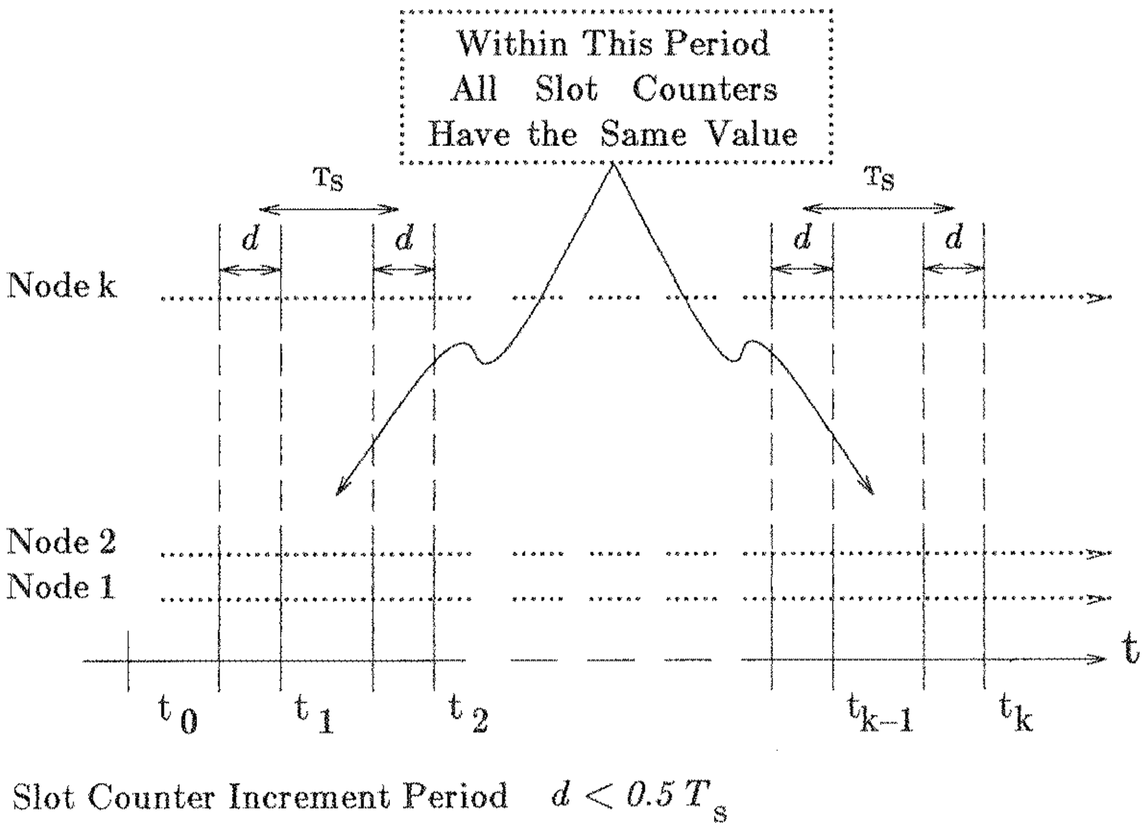


Figure 6.2: Global Synchronization Condition

In the discussion the following definitions are used:

Slot Phase – of a time slot is the first bit of the first control minislot.

Early Port – of a node is the port in which its slot phase occurs before the other port slot phase.

Later Port – of a node is the port in which its slot phase occurs after the other port slot phase.

Increment – of a slot counter follows the phase of the later port.

Time Stamp – of a packet which is sent via the early port is by next value of the slot counter, and of a packet which is sent via the later port is by the current value of the slot counter. Arriving packets are stamped in the same way: the packet from the early port gets the next value, and the packet from the later port gets the current value of the slot counter.

Net Cross Time – of a packet is the time it takes to arrive to all the nodes of the net. This time is measured by stamping the packet upon its transmission and upon its arriving. Thus, the net cross time is **exactly** f time slots.

Total Ordering Proposition:–

If the synchronization conditions SC1 and SC2 are maintained, and each data packet is time stamped by its originating node, then the global total ordering of packets is preserved.

Proof:–

The proof proceeds in three steps:

Step 1: Unique Time Stamping – following SC1 and the above definitions a packet which travels in the system is stamped uniquely upon transmission and arrival. This is a result of the unique pairing of phases on each node (SC1) and the fact that a packet crosses a net within **exactly** f time slots.

Step 2: Consistent Time Stamping – Assume that all the slot counters have the same initial value, then following SC2 they will continue to have the same value outside of the incrementing period d which is less than half a time slot, as shown in Figure 6.2.

Assume that: (i) a packet p_0 is sent from node i at t_0, (ii) a copy of p_0 is kept at node i, and (iii) this copy is time stamped every time slot by the slot counter of node i; then if p_0 is stamped whenever it reaches an arbitrary node j, then the two copies of p_0 will have the same time stamp. Thus, the two copies of p_0 are time consistent.

Step 3: Total Order – Let A_{t_0} be an arbitrary subset of packets which were originated and sent into the system at time t_0. After some k slots (k is any integer greater than zero) these packets are stamped again by their destination nodes. Clearly, since time stamping is unique and consistent all these packets will be stamped with the same value. Therefore, the relative time difference between any two arbitrary subsets A and B is preserved, and the total ordering is maintained.

6.3. Synchronization Analysis

The main emphasis in this section is on the design and analysis of a two–dimensional regular hypergraph. A partial hypergraph is discussed briefly and shown to operate similarly to the regular hypergraph. The objective of the analysis is to determine the necessary **operating conditions**, such that the synchronization conditions are maintained. The analysis is done first for a single net and then for the network.

In the following analysis the system is modeled as follows:

- Regular hypergraph – each node has two ports, and each net can be accessed via n ports.
- Each frame contains one slot (i.e., $T_f = T_s$). The analysis for $f > 1$ is very similar, and will be discussed in a later section.
- $R_{\max}$ is the maximum of all R's of the system's nets.
- One frame period (T_f) is exactly the time delay for the signal to travel twice the distance $R_{\max}$, i.e., $T_f = T_s = 2T_{R_{\max}}$.
- The time slot (T_s) interval, which is measured by a timer at each port, may have a maximum error of Δ_s. This error is the sum of placement error – the maximum error in measuring the port distance from the center of the optical star and clock error – the error of the node's local clock.

6.3.1. Net synchronization analysis

The Net's Synchronization Algorithm:–

Each node i determines the beginning of the next time slot by adding ($T_s - 2\Delta_i$) to the time it receives the first bit of the first control minislot (CMS). Figure 6.3 illustrates the correctness of this simple algorithm. Each node should delay the beginning of the next time slot by adding its round–trip delay to the circumference of the circle, which is exactly ($T_s - 2\Delta_i$).

The difference in determining the beginning of the next time slot among all the n ports of the net is $\pm\Delta_s$, so that the maximum timing difference is $2\Delta_s$. Since there is

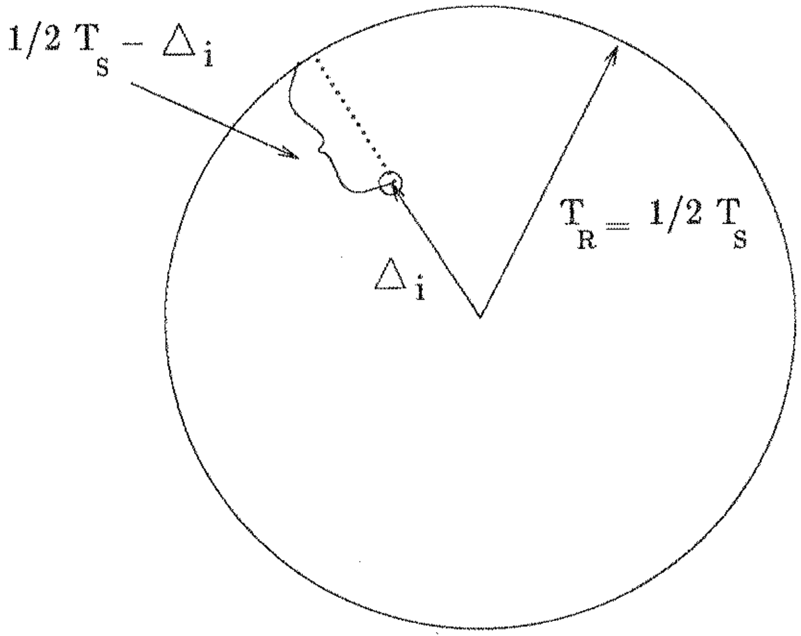


Figure 6.3: Net Synchronization

one slot in each frame, and at the beginning of each time slot all the net's ports are **resynchronized**, there is **no accumulated** timing error during successive time slots; i.e., the beginning of the next time slot is determined by the first bit of the current time slot.

First, the necessary time duration of each time slot is computed. Given T_{CMS} be the duration of each control minislot and T_{DMS} be the duration of the data minislot, what is the total duration of of time slot (T_s). Clearly, due to the error Δ_s there should be a safety margin between the transmission of two successive nodes. During each slot $r+1$ different ports are broadcasting. These ports are numbered according to the order in which they are broadcasting: {Port1, Port2, ... , Port(r+1)}, Port(r+1) sends the data packet during the DMS. Then, with respect to some global time reference $t = 0$, it is possible to determine when each port should transmit:

If	Port1 transmits at	t $= 0$,
then	Port2 transmits at	t $= 2\Delta_s + T_{CMS}$
:	:	:
and	Port(r) transmits at	t $= (r-1)(2\Delta_s + T_{CMS})$
and	Port(r+1) transmits at	t $= r(2\Delta_s + T_{CMS})$
and	then the next time slot will start at	

$$t = r(2\Delta_s + T_{CMS}) + 2\Delta_s + T_{DMS}$$

Hence, the slot duration should satisfy the following equality:

$$T_s = (r+1)2\Delta_s + rT_{CMS} + T_{DMS}$$

The above result on the duration of T_s is proven by induction on r.

The average efficiency ($\mu_{net-ave}$) of a single net is the fraction of each time slot used for transferring useful bits of information (data or control):

$$\mu_{net-ave} = \frac{rT_{CMS}+T_{DMS}}{(r+1)2\Delta_s+rT_{CMS}+T_{DMS}} = \frac{T_s-(r+1)2\Delta_s}{T_s}$$

The actual slot duration is the time interval between the first bit of the first CMSs in two successive slots. The actual length of T_s depends on the relative error between these two ports which is $\pm\Delta_s$. Hence, the time slot duration varies between $T_s-\Delta_s$ and $T_s+\Delta_s$. Therefore, the worst-case efficiency ($\mu_{net-min}$) is when all the net's ports have an error of $+\Delta_s$, and the efficiency is then

$$\mu_{net-min} = \frac{rT_{CMS}+T_{DMS}}{T_s+\Delta_s}$$

6.3.2. Global synchronization analysis

The basic mechanism for global synchronization is that a net which completes its time slot before other nets will delay the beginning of its next time slot, in order to match the other nets' time phases. Via the CMSs, the nodes indicate to their orthogonal nets how to match the phases.

The Distributed Global Synchronization Algorithm:–

(i) each node increments its slot counter according to the phase of the later port; (ii) each port indicates in its message the amount of time needed to match the phases of its two ports (only the port in which the time slot ended earlier will indicate a positive time delay); (iii) the additional delay information is incorporated only when it becomes **common knowledge** *by all the net's ports, and (iv) there are r different time interval corrections (opinions) for adjusting the beginning of the next time slot, and the ports use the* **longest** *one among these r corrections.*

The principle for achieving synchronization by taking the latest (or maximum) among different time phases is also used in [Lamp78]. The above global synchronization algorithm eliminates the resynchronization uncertainty, which could occur if a node would resynchronize before the timing information becomes **common knowledge** on the net. The global synchronization algorithm is used explicitly in the computation of the maximum timing difference; as a result the validity of this algorithm becomes apparent.

Each net is **resynchronized** by the first bit of the first control minislot. Then, for global synchronization analysis, the possible timing error in determining the beginning of the next time slot is $\pm\Delta_s$. For the **worst–case**, assuming that all the nodes of the system have the maximum timing error of $+\Delta_s$, the minimum global efficiency is

$$\mu_{global-min} = \frac{rT_{CMS} + T_{DMS}}{(r+1)2\Delta_s + rT_{CMS} + T_{DMS} + \Delta_s} = \frac{T_s - (r+1)2\Delta_s}{T_s + \Delta_s}$$

If a uniform error distribution is assumed, the average efficiency will still be close to the worst-case, since the global synchronization algorithm **forces** the nets to maximize their timing error.

The actual reduction in the system's efficiency in the worst-case is very small; in most practical situations it will be a few percent. For example, if $\Delta_s = 100\ bits$, $T_s = 8000\ bytes$, and r = 4, then $\mu_{global-min} = 98.28\%$. Note also that the efficiency does not depend on the number of nodes in the system, and that the timing error Δ_s can be much smaller.

Note that the part of the CMS which indicates how to match the phases of the two orthogonal nets causes an additional decrease in the network efficiency. Since this is very small, it is not included in the above expression.

6.3.3. Global timing differences

The maximum time difference should satisfy the local and global synchronization conditions. In the following analysis the worst phase difference between any two nets in the system is computed. Clearly, if this phase difference is less than half a time slot then SC1 holds (by its definition), and also SC2 holds since the phases of orthogonal nets are paired uniquely (SC1), so that the increment of all the system's slot counters cannot differ by more than half a time slot.

Worst-case scenarios – might bring about the worst timing difference in the system. There are two equivalent worst–case scenarios; the first one is shown in Figure 6.4. Under this scenario, (i) all the nodes of one net (say, **maxnet**) have the maximum timing error of $+\Delta_s$, (ii) all the other nodes have the minimum timing error $-\Delta_s$, and (iii) none of the nodes of **maxnet** transmit on the other nets for the maximum possible period of time, which is $l = \left\lceil \frac{n}{r} \right\rceil$ slots. In other words, **maxnet** does not tell the other nodes its differences for l time slots.

The second equivalent worst–case scenario has two orthogonal **maxnets**, and the rest of the nodes have minimum error. The nodes of the two **maxnets** do not transmit on their orthogonal nets for a maximum interval of l slots. These two scenarios are easily proved to be the worst, by systematically checking all possible scenarios.

Under this scenario for $f=1$, the maximum timing difference accumulated is

$$\Delta_{\max} = (\left\lceil \frac{n}{r} \right\rceil + 2)2\Delta_s .$$

In order to satisfy the synchronization conditions, the following inequality should be satisfied:

$$\Delta_{\max} < \frac{1}{2}T_s .$$

For example, if the net size is n=100 nodes (the system size is 10,000 nodes), and the other parameters have the same values as in the previous example, then $\Delta_{\max} = 5400\ bits$, which is much smaller than half the time slot size of 32,000 bits.

Hence, synchronization conditions for achieving total event ordering are feasible, even for very large systems.

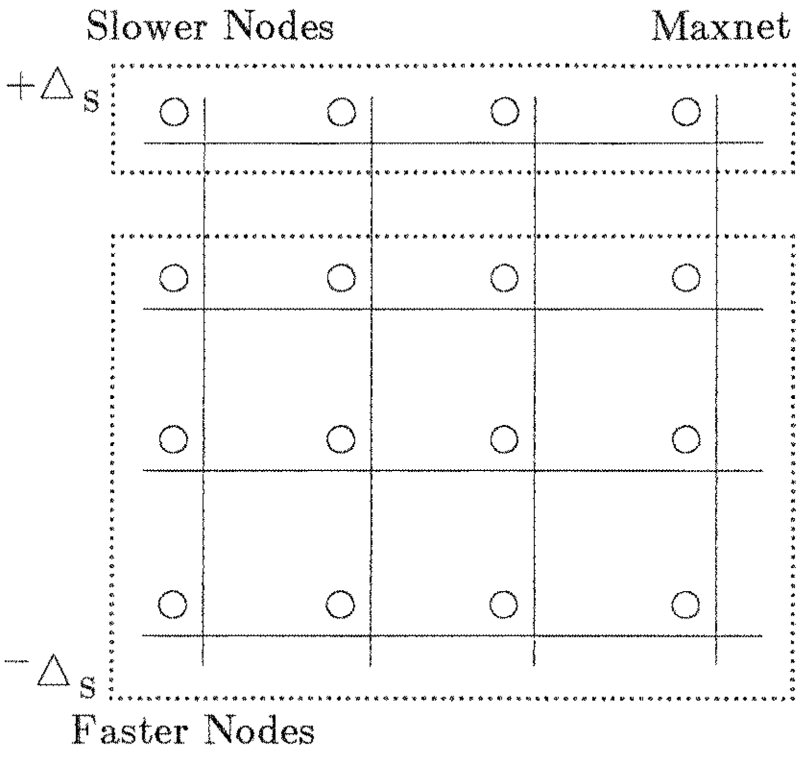


Figure 6.4: Maximum Timing Difference

6.3.4. Global initialization of the slot counters

The following are procedures for loading all the system's slot counters with the same value. These global initialization procedures can be done in a distributed or centralized manner. The following algorithms are for a 2D–R hypergraph.

6.3.4.1. Distributed initialization

This procedure is similar to the global synchronization algorithm. Each node selects at random an initial value for its slot counter. This value is transmitted by the control message during the CMS. When a node receives a slot counter value of another node, it adjusts this value by adding the appropriate propagation delay

(f +1 time slots), and then compares this value with the current value of its own slot counter. If the slot counter value is smaller, then it is replaced by the received value. Clearly, after $\left\lceil \frac{n}{r} \right\rceil + (f+1) + \left\lceil \frac{n}{r} \right\rceil + (f+1)$ time slots the largest among all the randomly selected slot counter values will reach all the nodes of the 2D–R hypergraph.

The above procedure is very fast (fraction of a millisecond); the problem is that the start–up phase of the system is much longer, i.e., the distributed initialization procedure is completed long before all the system's nodes become active. Therefore, it is reasonable to use a centralized algorithm when the system is initialized.

6.3.4.2. Centralized initialization

The following algorithm is centralized but can be performed by any of the system's nodes (the node can be selected at random). It has two phases (for a 2D–R hypergraph): phase 1 – a specific node broadcasts on one of its nets the message *"be ready to load t_0 into the slot counter"*. Then in phase 2 – the nodes which received this message broadcast the following message via their other port: *"load $t_0 + i$ into the slot counter"*. The number i indicates the number of slots passed between the original message (phase 1) and the time this message is actually transmitted. If n is the number of nodes on a net, then after at most $(n+1)$ time slots all the system's slot counters have the same value. Note that during this initialization procedure the synchronization conditions hold, and therefore, the value of i is consistent for all the nodes in the system.

6.3.5. Synchronization of a 2D partial hypergraph

In the following discussion the system topology is changed, such that each net has k nodes with 2 ports, and a nodes with one port $(k+a=n)$, as shown in Figure 2.3 and discussed in Section 2.2.

The efficiency results for the net and network synchronization remain the same as obtained before. The scenario for the worst–case timing difference is the same as for the regular hypergraph, but the consequences are slightly different. The nodes of **maxnet** cannot update the rest of the system's nodes within one time slot, i.e., the nodes with one port on nets which are parallel to **maxnet** have a distance of two nets from the nodes of **maxnet**. Now, if none of the k nodes (with two ports) of one of the nets, which is parallel to **maxnet**, will transmit before all the a nodes transmitted, then the worst case timing difference between **maxnet** and these a nodes is

$$\Delta_{max-part} = (\left\lceil \frac{n}{r} \right\rceil + \left\lceil \frac{a}{r} \right\rceil + 3)2\Delta_s .$$

Hence, in order to satisfy the synchronization conditions for the partial hypergraph, the following inequality should be satisfied

$$\Delta_{max-part} < \frac{1}{2} T_s .$$

6.4. Synchronization Analysis of a Larger Net

In the previous analysis it was assumed that each frame has exactly one slot. As the bandwidth increases, the physical length of the slot decreases, and the net can have more nodes in a larger area. If the number of bits in a slot remains the same, then every frame will have several slots, i.e., $T_f = 2T_R = fT_s$.

The synchronization conditions and algorithms remain the same. The consequences of this change can be on the synchronization efficiency and the global time difference. It is assumed that Δ_s in bit periods remains the same, even when the bandwidth increases.

6.4.1. Synchronization efficiency

The net and network synchronization efficiency is basically the same as in the previous analysis. This can be shown by assuming that each frame has $f\,(r+1)$ minislots, which implies that, for safety, $f\,(r+1)$[illegible]Δ_s[illegible]is added to the time frame. This safety margin is f times larger than before, but also the frame duration (in bit periods) is f times larger than before. Thus, the synchronization efficiency remains the same.

6.4.2. Global time difference

The worst–case scenarios remain the same, but there is additional time delay in order to match the phases of orthogonal nets. This additional time difference consists of: (i) f slots due to the delay of the node view on how to match the phases, and (ii) f slots due the delay until the phase match information becomes **common knowledge**. Thus, the maximum timing difference accumulated is

$$\Delta_{\max}(f > 1) = \left(\left\lceil \frac{n}{r} \right\rceil + 2f \right) 2\Delta_s .$$

In order to satisfy the synchronization conditions, the following inequality should be satisfied:

$$\Delta_{\max}(f > 1) < \frac{1}{2} T_s .$$

6.5. Discussion and Conclusions

The foregoing analysis exhibits the feasibility of achieving global event synchronization of a large area network with very high efficiency. This is due, first, to the use of a centralized, passive, optical star which can be simply synchronized; second, to the high bandwidth optical medium which can be time–shared by many ports, and third, to the two–dimensional hypergraph topology, which minimizes the distances between nodes. The limitation of global synchronization is that all events should be of the same size. The event size is one time slot, and for efficiency purposes should have a duration of several kilobytes.

Global synchronization can have an important impact on the overall complexity of the system's operation and computation control (see the following example on global mutual exclusion). Parallel distributed algorithms can use **common knowledge** as their basic principle. Since each event has its own time signature and the system is synchronized, these algorithms can be performed in an **open loop mode**, i.e., without the need for **acknowledgements.** This has two important consequences: (i) the execution of these distributed algorithms is efficient, and (ii) the system operation is simpler. There appears to be a tradeoff between a somewhat complex network operation with synchronization and the simplicity of the higher levels of system operation (communication and computation management).

6.5.1. Maximum clock synchronization

The synchronization procedure described above may be viewed as generating a slower synchronous clock from multiple asynchronous clocks. The maximum timing difference on a 2D–R hypergraph (with $f=1$) is $\Delta_{\max} = (\left\lceil \frac{n}{r} \right\rceil + 2)2\Delta_s$. This timing difference constitutes the maximum synchronous rate of the system's slot counters, which is the reciprocal value of the minimum slot period (basic event) in the system

$$SLOT\text{--}CNT_{\max} = \frac{1}{T_{s_{\min}}} = \frac{1}{4\left(\left\lceil \frac{n}{r} \right\rceil + 2\right)\Delta_s}.$$

Let $w = 4\left\lceil \frac{n}{r} \right\rceil$. The error Δ_s is a function of (i) the frequency differences among the high–speed transmitting clocks – $\pm b$ (typically less than 0.01%), and (ii) the physical measurement error $\pm d$.

$$\Delta_s = d + bT_s$$

Therefore,

$$T_{s_{\min}} = w(d + bT_{s_{\min}})$$

Thus,

$$T_{s_{\min}} = \frac{wd}{1 - wb}$$

Finally, the maximum synchronous clock is

$$SLOT\text{--}CNT_{\max} = \frac{1}{T_{s_{\min}}} = \frac{1 - wb}{wd} \approx \frac{1}{wd.}$$

Thus, the maximum system clock does not depend on the high speed clock differences but only on the physical measurement (d). The synchronization depends only on the number of nodes per net and not on the total number of nodes in the system. If the number of CMSs is increased, then the maximum system clock bandwidth is proportionally increased.

For example, let $n = 32$, $r = 4$, $d = 25$ nanoseconds, $w = 40$, and the maximum bandwidth is

$$SLOT\text{-}CNT_{\max} \approx \frac{1}{wd} = \frac{1}{1000} = 1 \; megahertz$$

6.5.2. Synchronous and asynchronous operation

A more general view of the synchronization procedure is possible. Each event may have multiple asynchronous subevents; therefore, the synchronization procedure can be viewed as placing an upper bound on the maximum timing difference among the asynchronous subevents.

One possible conclusion is that in order to have a more effective synchronization, the duration of each subevent should be longer, i.e., in the case of distributed computation, a larger subevent means a larger process granularity.

6.5.3. Distributed mutual exclusion

In order to illustrate the potential of global event synchronization and time stamping, a distributed mutual exclusion algorithm is presented in this section. They are designed for a regular hypergraph and use the CMSs or DMS for sending control messages. The mechanism discussed here allows the sharing of *independent, replicated items* among collections of distributed, cooperating processes. An *item* is a portion of a process (usually a data structure) that must be accessed atomically. For more details see [McOf87].

Net (or local) distributed mutual exclusion algorithm :– the node, wishing to acquire the control of a common resource or a shared item, broadcasts its request to all nodes of the net. A time stamp is attached to this message which is sent via a DMS. Since only one node can use the DMS during each time slot, no two messages on a net can have the same time stamp. Hence, the message with the earlier time stamp receives the resource control rights. Note that, since all messages are broadcast over the net, the mutual exclusion resolution is computed independently by each node, and no acknowledgement is needed.

Globally distributed mutual exclusion algorithm :– this algorithm uses the CMSs for sending control messages to all the system nodes and is performed in two steps.
First, the requesting node broadcasts the message *"node i wants x at time t_0"* (t_0 is the time stamp) to all its adjacent nodes via one port. Then these nodes rebroadcast the message via their adjacent nets. If these messages have the highest priority, then after $\left\lceil \frac{n}{r} \right\rceil + f + 1$ time slots, at most, the resource request message has reached all the network nodes. The requesting node can determine the validity of its mutual exclusion request by comparing the time stamp of its original request message with all other messages requesting this resource. In the case of two requests of the same resource originating at the same time, the procedure repeats after a random delay.

The case of having too many mutual exclusion requests at the same time is discussed in [McOf87]. One possible solution is a distributed protocol which limits the maximum number of requests on a net within a period of time of $l = \left\lceil \frac{n}{r} \right\rceil$ time slots.

In this case the first step lasts l slots and each node may try to reserve a space (by broadcast over the net) for its request which will be broadcast on all the orthogonal nets during the second step. If no space is available the node will request it again after l slots. The second step also lasts l slots, so the node should wait $2l$ time slots in order to determine its exclusive right.

CHAPTER 7.

TWO–DIMENSIONAL HYPERGRAPH

7.1. General

The physical layout of a 2D hypergraph is mapped onto a two dimensional square grid, as shown in Figure 7.1a, which is an example of a 5–by–5 2D regular hypergraph. For regular hypergraphs, each node is identified by an ordered pair (x,y):

$$node(x,y) \quad such\,that \quad 1 \le x \le n, \quad 1 \le y \le n$$

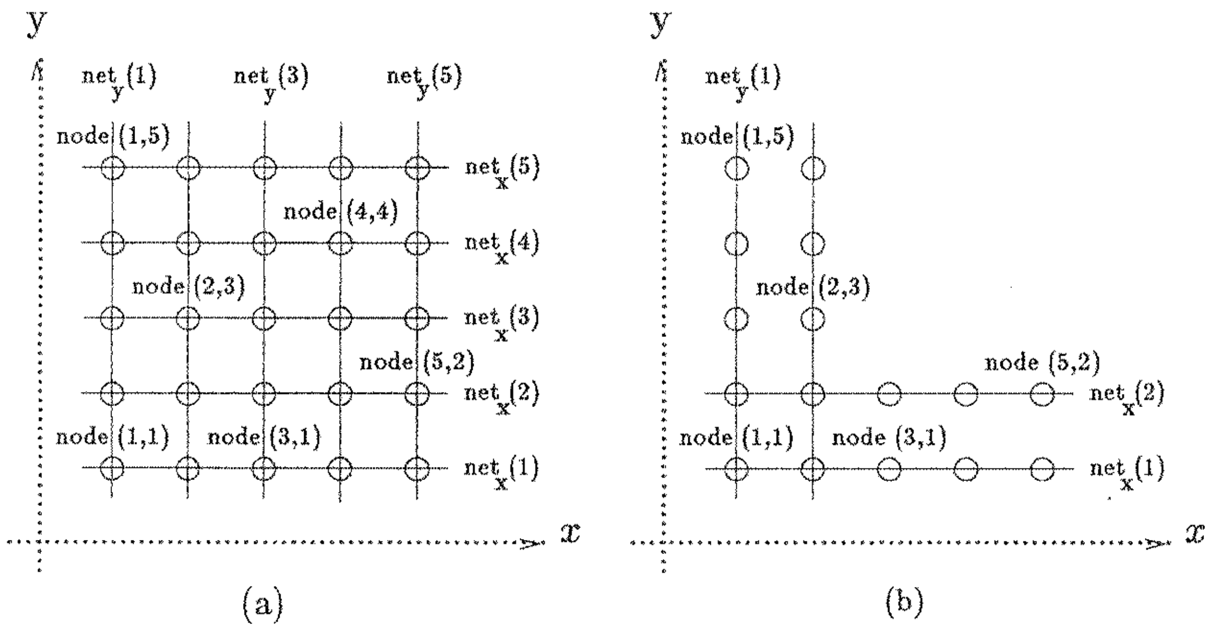


Figure 7.1: The Layout of a 2D Hypergraph

For a partial hypergraph, the nets in the x direction are indexed sequentially from 1 to k. The nets in the y direction are indexed sequentially from 1 to k. Each net has k nodes with two ports and a nodes with one port, such that $(n=k+a)$. A two-port node is identified by an ordered pair (x,y), a one-port node by either (x,∞) or (∞,y). Since the n nodes of each net are indistinguishable with respect to the center of the star, it is always possible to rearrange the network such that all nodes with two ports are within one square region close to the $x-y$ axis origin, as shown in Figure 7.1b.

In the discussion the following notation is used:

- $net_x(s)$ is the s^{th} net in the x direction. Thus, $node(r,s)$ is at the intersection of $net_x(s)$ and $net_y(r)$.
- **commonly known** refers to state information which has been broadcast from a known origin and has reached and been decoded by all the nodes of a net.
- $L_x(s)$ is the total communication load of $net_x(s)$, defined as the number of full buffers that are ready to be sent over the net. It includes only **commonly known** state information.
- $L_x(r,s)$ is the communication load of $node(r,s)$ via the $port_x(r,s)$ to $net_x(s)$; it includes only load information which is already **commonly known**.
- $L_y(r,s)$ is the load of $node(r,s)$ via the $port_y(r,s)$ to $net_y(r)$; it includes only load information which is already **commonly known**.

7.2. Two-Dimensional Node Interface

The node interface to a 2D network consists of two ports, buffers, a control unit, and a host interface, as shown in Figure 7.2. Each of the two ports is full-duplex and is designed as described in Chapter 4. The control unit is the heart of the node interface and includes the functions buffer manager, routing controller, and synchronizer.

The buffers at each node are organized as one uniform finite collection of m units. Each buffer can contain one packet of data. Any of these can be attached for read or write operations to the host bus interface, or $port_x$, or $port_y$. Each buffer can function as a FIFO or a RAM and, in the FIFO mode, it can be attached to two interfaces, for simultaneous read and write operations.

7.3. The Delay for Updating State Information

One of the design objectives is to minimize the delay of updating state information and thereby improving the distributed algorithm performance. The 2D regular and partial hypergraph are analyzed in this sectio, in order to find out the average– and worst–case propagation delay of state information in the system. It is assumed that each net has n nodes, and each slot has r control minislots. In the following discussion it is assumed that the state information should propagate via two nets in 2D–R hypergraph and via three nets in 2D–P hypergraph.

It is assumed that the control messages sent during the CMSs have a predefined structure, with a specific field (few bits) for each parameter. Therefore, a state parameter (e.g., load) which arrives at the node via $port_x$ can be transmitted via $port_y$ during its next CMS.

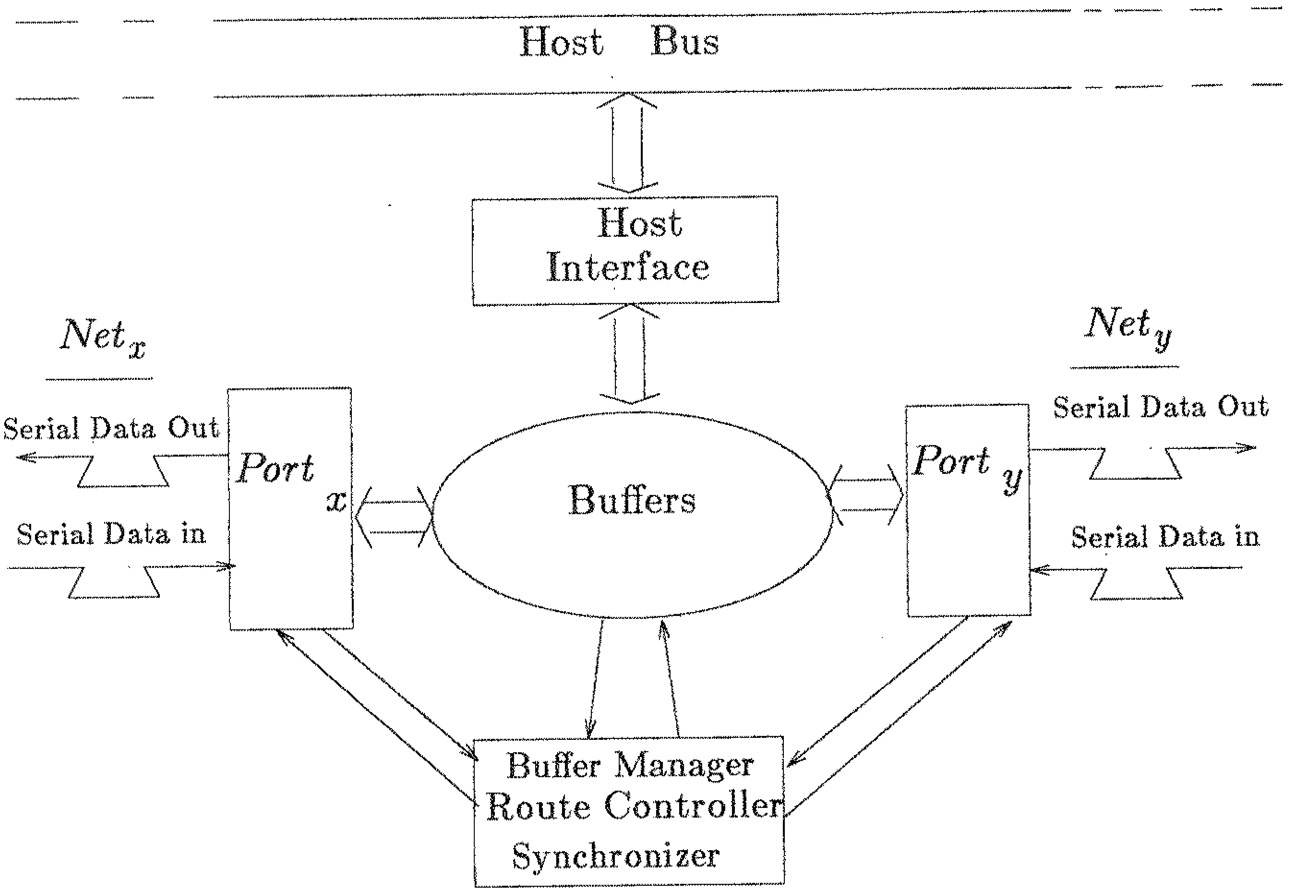


Figure 7.2: Functional Description of 2D Interface

7.3.1. The average delay

If a uniform use of the control minislots is assumed, then a node sends control information via its CMS every

$$l = \left\lceil \frac{n}{r} \right\rceil \ slots.$$

Thus, on average, new state information is sent with a delay of $\frac{1}{2}l$ and will reach to all the nodes on a net after an additional $f+1$ slots. The maximum distance between any two nodes of 2D–R is two nets. Therefore, the average delay for state information to reach all the 2D–R nodes is

$$t_{prop-2D-R} = 2(\frac{1}{2} l + f + 1) = l + 2(f + 1) \;\; slots.$$

In the 2D–P the information should propagate via three nets; therefore, the average delay is

$$t_{prop-2D-P} = 3(\frac{1}{2} l + f + 1) = \frac{3}{2} l + 3(f + 1) \;\; slots.$$

7.3.2. Upper–bound on the delay

In the worst case, the state information is transmitted after l slots; therefore, the upper–bound of the propagation delay through the system is $2l + 2(f+1)$ slots for 2D–R, and $3l + 3(f+1)$ slots for 2D–P.

The upper–bound parameter is important for the design of protocols that operate in an open loop mode, i.e., distributed algorithms which use time stamps as a substitute for explicit acknowledgements, as in the mutual exclusion example of Section 6.4.

7.3.3. Number of messages

The total number of control messages that are exchanged is an important criterion in the evaluation of a distributed algorithm. Since the system is globally synchronized by events, and assuming that each event is time stamped, then the order of the distributed events is preserved. Therefore, in many cases the control messages and their timing information can be sufficient for the distributed algorithm. One aspect of the complexity of distributed algorithms, which are based on **common knowledge**, can be measured by counting how many control messages are exchanged.

In order that a state parameter will reach all the system's nodes, the parameter is sent in a control message over one net and then over all the nets which are orthogonal to the original net. Thus, the number of control messages which are needed for propagating a certain state information (e.g., mutual exclusion request) to all the system's nodes is $n+1$ for 2D–R. For 2D–P the maximum distance is three, and a parameter propagates first via one net, then on k orthogonal nets, and then on $k-1$ parallel nets ($1+k+k-1 = 2k$ nets).

All the nodes of a net monitor the activity on the net in the same way and in a synchronous manner. Thus, the distributed activity can be measured by the total number of nets rather than by the total number of nodes. Therefore, the complexity of the synchronous hypergraph is twice the square root of the total number of nodes ($2\sqrt{total-nodes}$). This complexity is much lower than a point–to–point network, which is on the order of the total number of nodes. Thus, it exhibits the potential of the synchronous hypergraph for the implementation of complex distributed algorithms.

7.4. Routing Algorithms for a 2D Regular Network

The basic parameter for routing is the total load, which is **commonly known** on a net, rather than the total load of a node. An optimal routing algorithm will balance the load among all nets and thereby maximize the communication capacity of the system. The use of the net load as a parameter decreases the complexity of the routing algorithm relative to algorithms that use the load of each individual node. For example, the number of different parameters for an algorithm which considers global information of nets is only on the order of the $\sqrt{total-nodes}$.

Several methods for routing algorithms will be discussed in this section. These methods use the load information that is periodically exchanged by the control messages. One of the design objectives is to make these algorithms transparent to the host, i.e., the routing algorithms are performed by the network interface, independent of the system's software. Each node monitors its neighborhood (the activity on its two nets), and performs the routing algorithm without exchanging special messages with other nodes in the system.

The different routes on a 2D regular network may be classified into two types:

Primary route is the path with the least number of nets from one node to another. For the 2D–R hypergraph, the primary path has a length of one or two nets. For example, to get from $node(r,s)$ to $node(u,v)$ (which do not have a common net) there are two possible ways: (i) via $net_y(r)$ to $net_x(v)$, or (ii) via $net_x(s)$ to $net_y(u)$.

Secondary route is any route which is not a primary route. For the 2D–R hypergraph, a secondary route has a length of at least three nets.

7.4.1. The local balancing routing algorithm

This algorithm uses only the primary routes. Between every two nodes, in the plane of 2D–R, there are at most two primary routes. The routing algorithm determines which route to select. If the nodes have a common net the routing is via this net; otherwise, the criterion for selecting the primary route is by locally balancing the load of the two orthogonal nets, i.e., the data packet is sent via the net with the lesser load.

An active node is a node which uses its CMS, and by this asserts that "*I am Alive*." Let $node(r,s)$ be the source node, and $node(u,v)$ be the destination node, then there are four cases:

Case 1 – $r \neq u$ and $s \neq v$, and $node(r,v)$ and $node(u,s)$ are active.

If $L_x(s) \geq L_y(r)$, otherwise route is via $net_y(r)$ and $net_x(v)$, else the route is via $net_x(s)$ and $net_y(u)$.

Case 2 – $r = u$ or $s = v$, the source and destination have ports to the same net, the packet is sent directly via this net.

Case 3 – $r \neq u$ and $s \neq v$, and only one of the intermediate nodes, $node(r,v)$ or $node(u,s)$, is active, then the packet is routed via the active node.

Case 4 – both intermediate nodes, $node(r,v)$ and $node(u,s)$, are not active, then a secondary route is selected.

The implementation of local load balancing is simple. In order to compute the total load of a net, the nodes report the relative change in their load. Thus, the computation of the total load of one net is done by adding (or subtracting) the load changes.

7.4.2. The class of secondary routing algorithms

The class of secondary routing algorithms considers both primary and secondary routes. The objective of these algorithms is to balance the load over all the network's nets, and as a result to maximize the utilization and to minimize the average delay. With each control message, additional load parameters are transferred, which give information on the load of the orthogonal net. Via its two ports, the node monitors the activity on each of its two nets, and the information seen by $port_x$ is then transmitted over net_y, and vice versa.

In order to maximize the flow in the system and to minimize the average packet delay, the sum of the net's load along the route should be minimized.

Theorem: Maximum route length on 2D–R –

On 2D–R hypergraph, the secondary routing algorithm should not consider routes which are longer than three nets.

Proof:–

Let $node(r,s)$ be the source node and $node(u,v)$ be the destination node. The destination node can be accessed either via $net_x(v)$ or via $net_y(u)$. Assume that from some considerations $net_x(v)$ was selected; since the network is 2D–R hypergraph, a packet from the source node will intersect with $net_x(v)$ after traveling via one or two orthogonal nets, and the total length will not exceed three nets. The same argument is used if the $net_y(u)$ is selected. Thus, any route of length greater than three nets will just increase the sum of the loads along the route, Q.E.D.

In general, each port monitors the activity over its net and extracts three parameters:

(1) L – the current load on the net.

(2) L^{ave} – the average net load during the past h slots (h is to be determined by simulation of an actual system).

(3) U^{ave} – the average net utilization during the past h slots, which is the ratio between the full DMSs and h.

These three parameters are broadcast with the control messages via the other port onto the orthogonal net. As a result, all nodes receive these parameters from all the nets of the system. For maintaining global information, each node of the 2D–R should store $3(2n)=6n$ parameters, and for 2D–P network, only $6k$ parameters. Thus, maintaining global routing information for 2D partial network can be quite reasonable.

7.5. 2D Partial Hypergraph as a Centralized Switch

In this section the partial hypergraph is viewed as a centralized switching network, such that the nodes with two ports, constituting the switch, can be located in one building, and the nodes with one port are distributed within an area of a few kilometers around the switch.

Figure 7.3 is an example of a partial hypergraph of four nets (NET A, NET B, NET C, and NET D); each net is a passive optical star and is operated as described in Chapter 5. This is the same as the operation of the nets of a 2D regular hypergraph network. Two nodes of each net have two ports and are placed in one area, constituting the centralized switch. Note that this centralized arrangement is done for practical purposes (such as maintenance), and not because of different operational principles. The nets in Figure 7.3 can be viewed as local area networks, and the centralized switch is the means for merging them together. Thus, 2D–P hypergraph is a method for connecting local area networks (optical nets) together. This approach is significant, since it enables the construction of a small system which can be gradually extended.

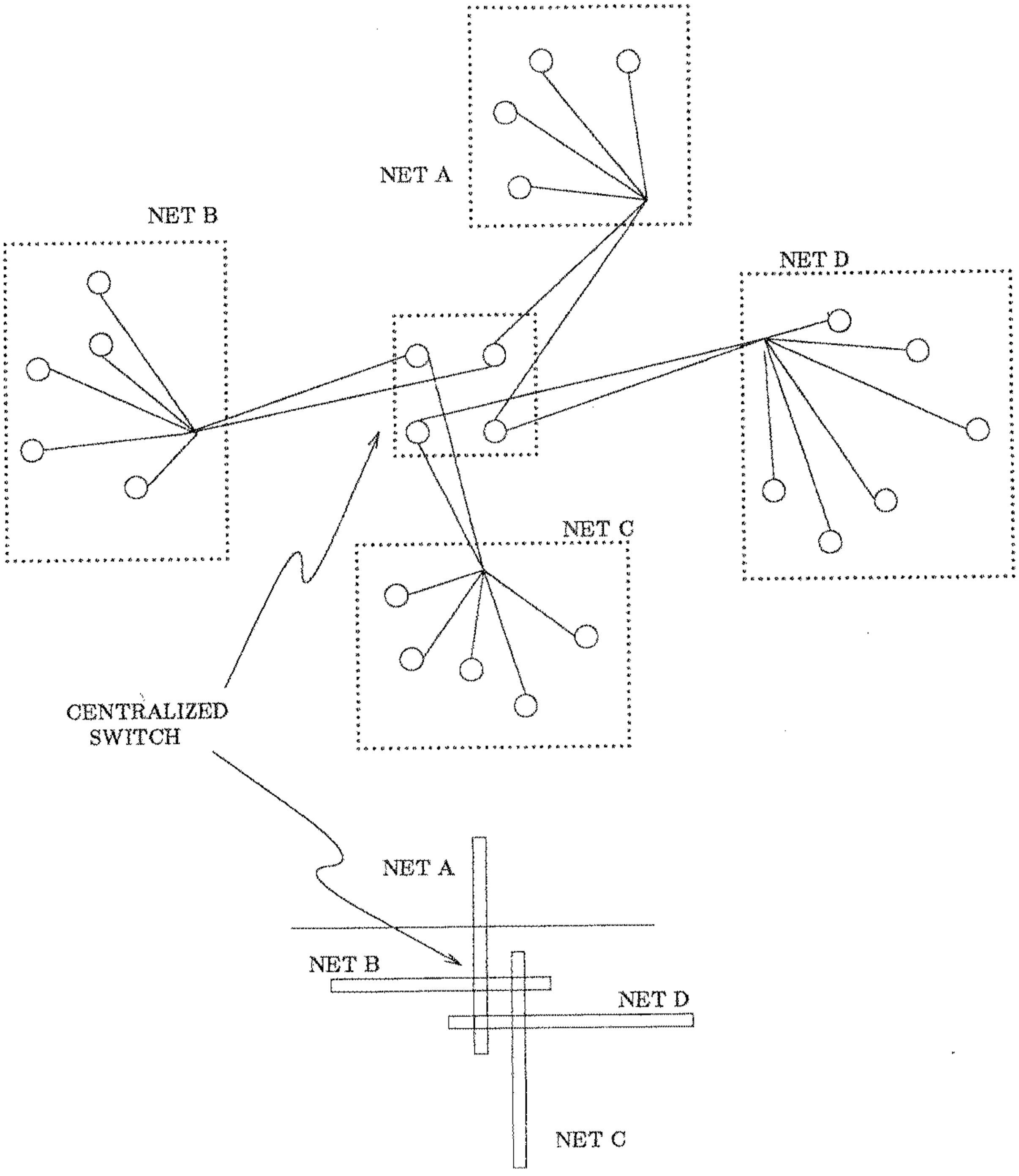


Figure 7.3: 2D–P Modeled as a Centralized Switch

In the analysis of a partial hypergraph as a centralized switch, three main issues are considered: (i) the bottlenecks and overflow prevention, (ii) conditions for ensuring maximum communication flow, and (iii) the effective bandwidth of a net.

7.5.1. Bottlenecks and overflow prevention

The nodes with two ports (the switch nodes) are potential bottlenecks, since a large portion of the communication traffic merges into them. Bottlenecks can occur under heavy load: i.e., when the number of full buffers which are **commonly known** is at least one. A bottleneck may result in buffer overflow and the loss of packets. In the following discussion it will be shown how overflows can be prevented.

In order to achieve these objectives the following conditions are determined, and then a theorem is proved.

Buffers Condition 1:

At a switch node, the set of buffers ($\{B_1, B_2, B_3, \dots , B_m\}$) is divided such that no more than $\left\lfloor \frac{m}{2} \right\rfloor$ can be in the queue of $port_x$, and no more than $\left\lfloor \frac{m}{2} \right\rfloor$ can be in the queue of $port_y$.

Periodic Exchange Condition:

Each switch node can broadcast state information via its two ports using the control messages during its CMS, at most every l' time slots.

Overflow Hazard Flag Condition:

The control message has a flag which indicates overflow hazard. When a port turns the overflow hazard flag on, the other nodes on this net will not transmit any more packets to it. The maximum latency or delay for turning on the overflow hazard flag, and for this information to get to all the nodes on the net, is

$$HAZARD-FLAG_{delay} = l' + f + 1,$$

which is the time until the next CMS, plus the time for this information to propagate through the net.

Buffers Condition 2:

The total number of buffers at each switch node is greater than $2l' + 4f + 2$, or that

$$\left\lfloor \frac{m}{2} \right\rfloor > l' + f + 1 + f,$$

the expression $l' + 2f + 1$ is the maximum number of packets which can arrive at that port from the time it has been decided locally (at the node interface), until the time in which this state information is **commonly known** on the net. Note that the second f is for the possible f packets which can be on their way, from the time the new state is **commonly known**.

The Algorithm for Turning On the Overflow Hazard Flag:

A switch node will turn on the overflow hazard flag when it broadcasts via $port_x$ over net_x, if the queue of full buffers ready to be sent via $port_y$ over net_y is greater or equal to $\left\lfloor \frac{m}{2} \right\rfloor - l' - 2f - 1$, and similarly for the overflow hazard flag which is broadcast over net_y.

Theorem:

The above conditions and algorithm are sufficient for preventing overflow.

Proof:

Whenever any of the two overflow hazard flags are turned on, the switch node can control the traffic which is generated locally by its host, by preventing the host from filling up buffers. Assume that, in the worst case, all the traffic from one net should be switched to the other net. Thus, no more than $l' + 2f + 1$ packets can arrive from the moment the hazard flag is locally turned on until this state is effective at all of the net's nodes. Therefore, using the above algorithm is sufficient to prevent overflow.

7.5.2. Ensuring maximum flow

The use of the overflow hazard flag may cause reduction in the performance of the system, i.e., the above conditions can reduce the effective communication capacity of the network. To prevent this additional condition is introduced:

Maximum Flow Condition:

When an overflow hazard flag is turned on over net_x, there should be a minimum number of full buffers in the queue to net_y, such that this queue will not become empty before the overflow hazard will be turn off. This minimum number is $l' + 2f + 1$, and the following inequality holds:

$$\left\lfloor \frac{m}{2} \right\rfloor - l' - 2f - 1 > l' + 2f + 1 \quad or \quad \left\lfloor \frac{m}{2} \right\rfloor > 2(l' + 2f + 1).$$

Theorem:

The above condition is sufficient to ensure maximum flow.

Proof:

Clearly, a reduction may happen if the queue of $port_x$ is empty, while the traffic over net_y to this node is halted by the overflow hazard flag. The time for turning off the overflow hazard flag is $l' + f + 1$ slots, and additional f slots are required for a packet to arrive at this node. Thus, $l' + 2f + 1$ full buffers ensure that the node will be able to send packets continuously.

The value of l' determines the total number of buffers in the interface. Therefore, in order to reduce this number, the ports of a switch node should have more access to a CMS, i.e., the switch node should have priority over other nodes in the use of the CMS. This will also improve the response time of the network.

7.5.3. Effective bandwidth

In this analysis, the system is modeled in order to find the number of nets required for constructing a system with some given parameters. In the following model the system is homogeneous; i.e., all the nodes are identical in their behavior, and all the nets have the same number of nodes.

The following parameters are defined:

- n_{net} – the number of nodes on each net ($n_{net} = k + a$).
- n_{sys} – the total number of nodes in the system $n_{sys} = n_{net}^2 - a^2 = k^2 + 2ka$.
- BW_{node} – the node's bandwidth, or the *average* traffic which the host of each node generates (it is the same for nodes with one or two ports).
- BW_{net} – the net's bandwidth, the *maximum* traffic which the net can transfer.
- b – the ratio between BW_{net} and BW_{node} $b = BW_{net} / BW_{node}$.

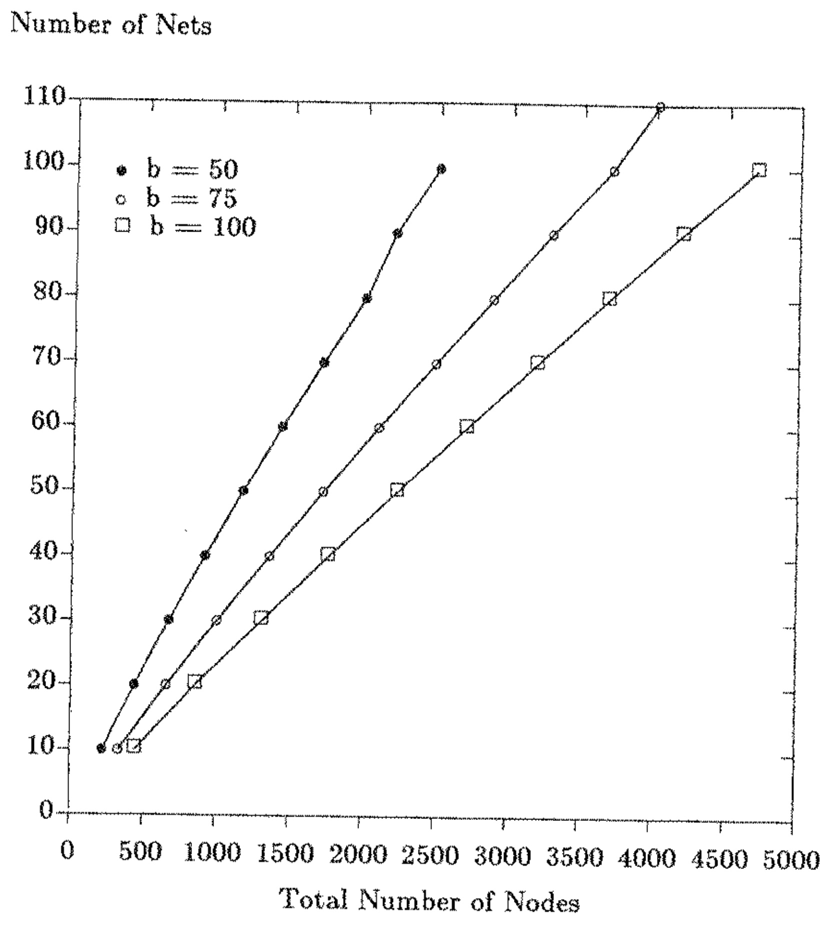


Figure 7.4: The Number of Nets as a Function of the Total Number of Nodes (b is fixed)

- d_{ave} – the average distance between two nodes of the network (measured in the number of different nets in which a message should travel). In the 2D–P network the possible distances are 1, 2 or 3 nets.

In order to support the average communication in the system, the following inequality should be satisfied:

$$2kBW_{net} \geq n_{sys} BW_{node} d_{ave},$$

i.e., the total communication capacity should be greater than or equal to the total traffic generated by the system's nodes, which should be multiplied by the average distance (number of nets) each packet travels.

The average distance in the homogeneous 2D–P is

$$d_{ave} = \frac{2ka\,[1(k+a-1)+2k\,(k+a-1)+3a\,(k-1)] \;+\; k^2[1(2k+2a-1)+2((k+a-1)^2-a^2)]}{n_{sys}^{\;2}}$$

Using the above expressions, it is possible to show various relationships among the different parameters (see Figures 7.4 and 7.5). The parameter b represents the

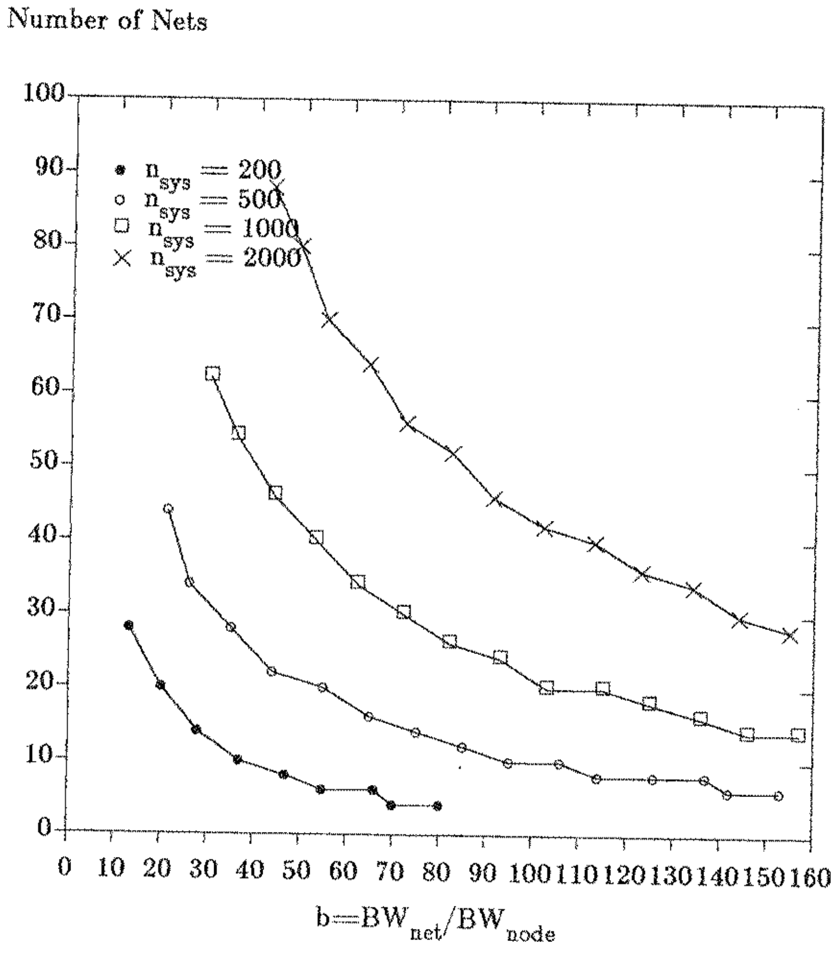


Figure 7.5: The Number of Nets as a Function of the Ratio $\frac{b = BW_{net}}{BW_{node}}$

state of a *given technology,* and one can assume that its value will increase in the future, with the result that a single net will be able to support the communication of more or busier nodes. For a larger value of b and a in a fixed size system, the number of nets is smaller. Also for a fixed value of b, as the system size enlarges, the net size enlarges as well. It is also apparent that the number of ports on each net of the partial network is larger than $\frac{b}{2}$ but smaller than b.

CHAPTER 8.

FAULT TOLERANCE OF THE OPTICAL HYPERGRAPH

The basic principles which enable this optical architecture to tolerate physical failures are outlined in this chapter. The scope of the fault tolerant (FT) enhancements includes functions performed by the network interface and not FT functions performed by the hosts. Furthermore, the realization of some FT functions by the interface reduces the complexity of the distributed software.

The basic architectural and operational principles of the system are very suitable for fault tolerant enhancements. The two most important properties are

(1) **Passive optical star** – this centralized topology is extremely suitable for fault tolerant architecture because

(i) The net can tolerate multiple link and/or coupler failures (i.e., graceful degradation of performance).

(ii) The broadcast transmission over the net is redundant in nature. As a result, all messages are received by all the net's ports. In general, much of the state and time information, received by each node during the CMSs, is redundant.

(iii) The net has a single time reference, which enables distributed net synchronization and well–defined state transitions at the end of every time slot.

(2) **Periodic exchange of state information** – this mechanism is equivalent to exchanging "*I am alive*" messages in a synchronous and deterministic manner, and any deviation from this behavior signals a failure. Note that the actual

information being exchanged determines what failures can be tolerated.

Fault tolerance (FT) is the surviving attribute of a system. The following discussion emphasizes how the system can tolerate physical failures rather than man–made faults; for FT taxonomy see [Aviz78]. The system consists of multiple computing and communicating nodes the objective being that a failure of some nodes will not disable the whole system. Thus, the basic FT methodology, a graceful degradation of performance, is completely distributed; i.e., no "*king*" is used in the fault tolerant procedures. As a result, the FT mechanism exhibits the equivalence between a regular and partial hypergraph; i.e., the regular hypergraph can be gracefully degraded in a distributed manner into almost any arbitrary partial network.

8.1. The Fault Tolerance Objectives

There are two major reasons for making this system capable of tolerating failures:

(1) Global Event Synchronization, in which the fundamental attribute of this system makes the system more vulnerable. A single synchronization failure may cause an entire system to fail. It will be shown that this unique optical architecture is, indeed, capable of overcoming multiple synchronization failures in a complete distributed manner.

(2) Lossless, the decreased probability of a **packet loss** whereby communication reliability is increased. A packet, transferred to the network interface, would not be lost without leaving some traces. A successful data transfer occurs when (i) the packet has reached its destination and left the network interface, or (ii) the packet was detected as faulty by the **sending side** (self check by each port) and

then retransmitted by the interface, or (iii) the packet can not reach its destination because of some port failures, and a message is then returned to the packet's origin. A loss of packet occurs when the packet is transferred to the network interface and then **"disappears"** without any **direct** trace. Only a higher-level communication protocol, at the destination host, can indirectly detect a missing packet and then send a special retransmission request.

8.2. Methodology

In general, a fault-tolerant procedure may have three steps: (i) fault detection – achieved by adding redundant parameters (e.g., extra bits in the broadcast control messages), (ii) fault diagnosis – determining the source of the failure, and (iii) fault recovery – continuation of the system's regular operation. In the following analysis the emphasis is on the individual net. Once a net is FT and the connection between the two ports of the interface is made FT, the whole system can then be made FT.

The method for fault-tolerant of the global event synchronization and time-stamping is by using fault masking; i.e., majority voting in real-time. This method uses the inherent redundant nature of broadcasting over the optical star.

The fault-tolerant methodology for preventing packet loss is based on the ability to detect faulty ports within a very small delay. Once a faulty port has been detected, the net gracefully degrades its operation by isolating and bipassing the faulty port from the net.

The FT analysis proceeds **bottom-up**, as shown in Figure 8.1. The lower level and the basis of the FT analysis is the transmission medium (optical star). The medium is analyzed with respect to physical disconnection and for external interference (noise).

The basic FT principle of the system is "**isolate and skip.**" A port which diagnoses itself as faulty will **isolate** itself from the net. Due to the deterministic, periodic exchange mechanism, all other ports can detect idle or faulty ports and **skip** them. The last step models some possible failures in the synchronization and packet transfer operations and then presents solutions for overcoming them. These procedures are performed in real time, in a distributed manner.

8.3. Optical Medium Failure

The basic fault–tolerant features of the net are derived from its centralized, passive, optical star topology, which can tolerate multiple link failure.

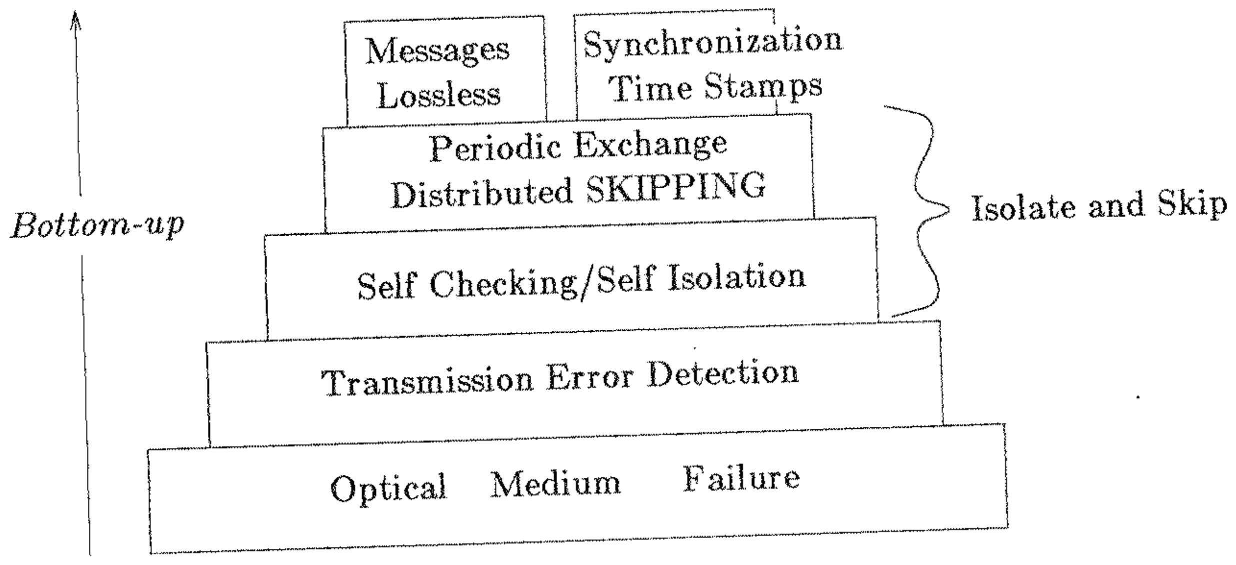


Figure 8.1: Fault Tolerance Methodology

Fault model of the medium – the cause of failure of the optical medium can be as a result of a mechanical damage, which results in a disconnection of the medium or star coupler. The optical star can be protected by a metal box in order to prevent physical damage. The optical net cannot be completely disconnected by a single coupler failure. If the star is constructed, as shown in Figure 2.5, by a $\frac{n}{2}\log_2 n$ couplers, then at least $\frac{n}{2}$ couplers should fail in order that the whole passive star will fail, or that the net will become completely disconnected. Any partial failure where the net is still partially connected, can be overcome by the **isolate and skip** mechanism. Similarly, multiple fiber links failures are overcome by the **isolate and skip** procedure (unlike in rings or linear buses where a single broken link can bring down the whole net).

Furthermore, the optical medium is immune to almost all electro–magnetic, magnetic, and electric interferences (EMI, RFI, etc.), which sharply reduce the probability of intermittent failures, i.e., bit failures in the optical channel. The electronic circuits, not the optical medium, is the main source of failure.

8.4. Transmission and Error Detection

The next step in this FT design is the detection of errors in the serial bit stream (methods for error corrections are not discussed in this chapter). The detection is done by the digital part of the receiver interface. Thus, detected failures can occur in the optical medium, in the analog circuitry of the transmitter, or in the receiver.

The serial bit stream is encoded by the conservative code, which is self–clocking; under normal conditions, the serial–to–parallel conversion is free of the metastable problem, which reduces the reliability of the digital receiver.

Some fault models in the serial bit stream and their corresponding FT solutions are

(1) A faulty detection of the codeword's delimiting transition – this might happen as result of a missing transitions (rising and falling edges); as a result, the sampling of the current codeword is extended into the next codeword, so that all the following codeword samplings are not done on the exact codeword boundaries.
The solution is to map all the unused codewords to an error signal; e.g., the 8B/12B conservative code has 462 possible codewords; since only 256 are needed, the remaining 206 codewords can be used for error detection. Thus, after a few erroneous samplings of codewords, a codeword which is mapped to the error signal would occur. If the number of codewords which are used for legal data word is p, and the total different possible codewords is s, then with probability $\left(\frac{p}{s}\right)^n$ after sampling n successive codewords, the error signal would be asserted.

(2) A faulty transition position – the number of transitions is still preserved, but one of the codeword edges has been shifted.
The previous solution will work only with a probability of $\frac{p}{s}$. Thus, for this failure of the conservative code, the CRC error detection code can be used. The CRC can be computed for each packet before the conservative transformation, and its reminder is added to the tail of the packet. At the receiving side, after the conservative decoding, the CRC is computed and checked.

(3) Collisions – the primary reason for a collision is a failure in the distributed synchronization procedure at one of the net's nodes. Since the transmission from all

the nodes merges to one point in space, all the nodes are capable of detecting collisions of any length.

Collision is detected by using the conservative/balanced code, as described in Section 3.3. Each node can determine if its own transmission was involved in the collision if it knows the collision time. This information can then be used in the self–isolation procedure and in the synchronization, fault tolerant procedure.

8.5. The Interface and Self Isolation

In the network interface, described in Chapter 4, each port is full duplex. For FT purposes it is assumed that the operation of the receiver and transmitter sides are completely independent. The interface is designed hierarchically, and only very small parts of its digital components change their state in a high bandwidth. The full duplex, hierarchical interface and the synchronous decoding improve the reliability of the interface.

From the net point of view, the objective of the FT procedure of the port is to stop the transmission in case of a failure; i.e., self isolation, in which the port does not use either its CMS or any previously reserved DMS. This strategy has major characteristics: (i) it enables the net to maintain or regain synchronization among the nonfaulty nodes, and (ii) it prevents the propagation of error in the system. The isolated port can be passive (not necessarily dead), and continue to monitor the activity over the net.

The following are several fault models which result in self isolation. The means (hardware or software) for detecting these failures is explicitly described. In the context of this discussion there is no attempt to cover all possible failure modes. It is assumed that the conservative/balanced code is used, which makes it possible to determine if a transmission failure is because of a collision or other causes.

(1) Transmission failure – the receiver side of the port receives its own transmission after it was propagated via the optical star. If an error (not a collision) is detected, then the port can retransmit the packet. Since a packet propagates through the net within f time slots, then after $f+1$ slots the port can determine if the transmission was successful and only then discard the transmitted buffer. If the node failed for two successive transmissions, then the node concludes that something is faulty and is isolated from the net.

(2) Transmitter clock failure – this clock is used for shifting the serial data out, and for determining when each slot begins. A small deviation of this clock frequency can be catastrophic for the net operation, since it can cause collisions.
A redundant clock, functioning as a watch–dog timer, detects clock failures. Any disagreement between these two clocks causes the self–isolation of the port.

(3) Collision – a port which detects a collision diagnose to check if its own transmission was involved. It is done by measuring the time difference between the collision detection and the port's last transmission. If the port's transmission is involved in the collision, the port immediately isolates itself.

A self–isolated port performs self–tests in order to further diagnose the source of the failure. If nothing has been found, the port will become active again by simply starting to send control messages during its predetermined CMS.

8.6. Skipping over Self–Isolated Ports

The net continues its operation in the presence of self–isolated ports by skipping or bipassing them during the CMSs and DMSs. This graceful degradation mechanism enables the active ports to continue normal operation. The use of the CMSs is deterministic, and the order of use is predetermined and commonly known. Hence, if a port is idle during its CMS, its successor will continue in its turn. If a port is idle in using its CMS twice successively, then (i) all the DMSs which are reserved by the port are deleted, and (ii) no data packet is sent to this port.

Since there is a one–time reference on each net, the idle information about the self–isolated ports is incorporated into the state transitions of all the nodes of a single net at the same time (a well–defined state transition), which greatly simplifies the handling of idle nodes. Whenever a port rejoins the activity on the net, by using its control minislot, the other ports stop their skipping.

8.6.1. Hardware requirements for skipping

The hardware requirement for skipping is measured by the number of registers, which contain the addresses of the nodes transmitted before this node. Each node has its own set of registers.

In order to handle any number of failures in the use of the control minislot, each node should have a complete list of the nodes in its subset. In the uniform case of n nodes on a net, which use r CMSs, then each node should have a list of

$$l = \left\lceil \frac{n}{r} \right\rceil \textit{ registers.}$$

The use of the DMS is dynamic and depends on the specific scheduling algorithm. If a uniform round–robin algorithm is used, then several cases can be identified:

Case 1 – no skipping capability, each node which uses the DMS should store the addresses of $f+1$ nodes which transmit in the sequence before it.

Case 2 – in order to overcome a single failure by skipping, the node should store the address of one more node in the sequence which transmits before it, or total of $f+2$ addresses.

Case 3 – in order to overcome g failures by skipping, the node should store the address of g more nodes in the sequence which transmits before it, or total of $f+1+g$ addresses.

Case 4 – in order to overcome any number of failures by skipping, the node should be able to store the address of all the $n-1$ nodes of the net.

8.7. Lossless System

One of the major objectives of making this system fault tolerant is to minimize the probability of a packet loss. It is achieved by minimizing two events:

(i) sending a packet to a faulty port, achieved by using the complementary acknowledgement mechanism, and

(ii) sending a packet to a port with no empty buffers, achieved by using overflow hazard flags in the control messages.

8.7.1. Complementary acknowledgement

The use of the control minislots by the net's ports is deterministic and periodic, and can be viewed as "*I am alive*" messages; i.e., a port may check itself continuously, as described before, and as long as everything is normal, this port will continue to use its CMS. Once a failure is detected, the port stops using its CMS, and thereby indirectly notifies all the other ports of its failure.

This technique can be viewed as a **complementary acknowledgement**. There are two advantages of this technique; first, communication validity is close to what can be achieved with the usual acknowledgement protocol, and second, the significant acknowledgement overhead is saved.

8.7.2. Overflow prevention

Each node has a finite set of m buffers; one of the objectives of the buffer controller is to prevent overflow and to ensure maximum utilization of the communication capacity. Section 7.5 shows how a switch node of the 2D–P hypergraph prevents overflow and ensures maximum flow. All the nodes of 2D–R can be viewed as switch nodes, and the conditions and algorithms for achieving these objectives are the same. Thus, if the total number of buffers is m, the queue of each port can not have more than half of them. Each node can transmit a control message every l' slots, and f is the number of slots in each frame, then the necessary condition for preventing overflow and ensuring maximum flow

$$\left\lfloor \frac{m}{2} \right\rfloor > 2(l' + 2f + 1).$$

8.7.3. Verifying successful transmission

Two more conditions are needed to be added in order to verify with a high probability that the transmission of a packet is successful:

(i) A port performs a self check on every control message it sends during the CMS; i.e., the port waits $f+1$ time slots and compares the receiving control message with the original one. If the comparison fails, then the port will try to retransmit once more. If it fails again, then the port will isolate itself and the packet will be sent to the host with an unsuccessful transfer message.

(ii) A node does not discard a message for $\left\lceil \frac{n}{r} \right\rceil + 2f + 1$ time slots. Within this period of time, the destination node should transmit via its CMS. If the node fails to do so, then the source node should assume that the node is faulty and that the message failed to reach its destination. In the case of failure, the port interface will notify its host.

The delay period for these two conditions is taken in parallel, and since the time for the second condition is always longer, it is necessary to wait only that period of time.

8.8. Synchronization Fault Tolerance

The global event synchronization is a fundamental operation principle of this system. A failure of the synchronization mechanism on one of the net's ports can cause the failure of the whole net. It important to ensure that a net failure will not cause failure of other nets.

8.8.1. Synchronization, failure detection, and protection

The optical net is a shared medium; thus, a synchronization failure is very likely to be manifested by a collision. A conservative/balanced code detects a collision. In order that a node will determine if its own transmission was involved, the node measures the time between its transmission and the collision detection. It is done at the receiving side of the port, by an independent clock. If the collision was detected, within a delay of f slots to $f+1$ slots, since the port's last transmission, the port will then assume that its transmission was involved in the collision. As a protection measure, the ports which were involved in a collision will isolate themselves.

8.8.2. Slot counter failure

The objective of the global synchronization procedure is to uniquely time stamp uniquely all computation and communication events. Thus, a slot counter failure will cause a false time stamp, which can result in a faulty computation.

This type of failure can be masked by comparing the value of the slot counter with the time stamps of the control and data messages, which are continuously received by the node. In the case of disagreement, the node performs majority voting and updates the value slot counter. If this failure repeats, the slot counter should be declared faulty, and the two ports of the node will isolate themselves.

8.8.3. Global synchronization error detection

Error detection in the global synchronization algorithm is achieved by measuring the timing difference between the two ports; if it exceeds $\frac{1}{2}t_s$ an error would be indicated. Hence, if a port causes an arbitrarily long delay (as a result of malfunction), it can be detected by all the other ports.

Note, that this type of error is not likely to happen, since an error should be detected on one of the system's nets (usually a collision).

8.9. Discussion

In the previous fault–tolerance discussion, the main redundancy technique used was hardware. Other FT techniques such as (i) communication redundancy – extra redundant messages, or (ii) time redundancy – repeated transactions, is used to a very limited extent. Thus, fault tolerance enhancements minimally increase the system's operational complexity.

The complementary acknowledgement method ("*I am Alive*"), and the ability to execute a distributed transaction in an *open mode* fashion (see Section 6.4) are important for FT and for reducing the complexity of the distributed computation in the system.

The discussion on fault tolerance in this chapter added another dimension to the significant capabilities of the proposed optical hypergraph architecture. It is very important to note that both the synchronization and fault tolerance mechanisms are derived from the centralized, passive, optical star topology, and the periodic exchange of state information. Thus, the topology and the periodic exchange are important building blocks of large distributed systems.

CHAPTER 9.

INTEGRATION OF VOICE ON A SYNCHRONOUS HYPERGRAPH

9.1. Introduction

The integration of real–time communication with digital data communication is of growing interest (see [FiTo86], [Grub81], and [Mont83]), for several reasons: Networks' bandwidths are growing rapidly, so the excess capacity can be used for telephony or video, which saves the construction of separate networks. Digitized voice telephony has better quality than the analog one. At present, and even more so in the future, there will be a demand for an off–line digital recording over a network of voice and video for future use, so real–time data transfers over a large network are an actual necessity. In this chapter it will be shown that the integration of voice on an optical hypergraph architecture is more efficient and more effective as the net bandwidth increases.

In general, real–time communication constraints (the necessity to complete the communication in a bounded time) introduce additional complexity to communication management. However, real–time communication on a synchronous hypergraph network introduces little additional complexity and is an immediate result of global event synchronization.

The voice transmission is packetized and sent over the net by constructing virtual multiplexers/demultiplexers. It is done without changing any of the previous design elements. Each telephone conversation has a very low bandwidth. Therefore, it is important to be able to manage and transfer efficiently small blocks of data.

The following discussion presents the basic protocols for integrating voice into the network. The scope of the discussion is limited to the principles of operation and to the analysis of the integration. This discussion is limited to two–party conversations.

9.2. Telephony Communication Characteristics

A typical sampling rate for voice communication is every 125 μseconds or 8,000 samples per second. Analog samples are converted into 8–bit bytes, which are grouped into voice–parcels. The size of a typical voice–parcel is between 160 and 400 bytes of speech, corresponding to 20–50 milliseconds of speech. The duration of a typical telephone conversation is about 200 seconds (3 minutes).

The following parameters will be used:

- T_p – the duration of speech which is sampled into one parcel,
- BW_{net} – the bandwidth of a single net,
- T_{DMS} – the duration of one data minislot,
- T_{PAR} – the transmitting duration of one voice–parcel.

9.3. Principles of Voice Integration

The integration of voice into the system is achieved by adding the following features to the optical architecture:

(1) Cycle – the time is divided into cycles of c slots each. The duration of each cycle is exactly T_p, the time which one voice–parcel is generated. The number of time slots in every cycle is $c = \dfrac{T_p}{T_{PAR}}$.

(2) Reservation Register – each node has a slot reservation register of c cells, with 2 bits per cell, defined as follows:

00 – the slot is free

01 – the slot is reserved by another node

10 – the slot is mine

11 – the slot is reserved for data communication

Free slots can be reserved for voice communication or can be used for data communication. Slots reserved for data communication cannot be reserved for voice. Note that the access control mechanism continues to operate as previously described, but it uses only the free slots and slots which have been reserved for data communication.

(3) Virtual Multiplexing – each data minislot (DMS) is subdivided such that each subsection will contain one voice–parcel. The number of voice–parcels in every DMS is $p = \frac{T_{DMS}}{T_{PAR}}$. These p parcels are sent from one origin but can have different destinations on that same net.

(4) Virtual Demultiplexing – each port of the net, which receives the broadcast packet with the p voice–parcels, extracts those which are sent to it and discards the rest. It is done on–line by checking an 8–bit logical address header of each parcel.

(5) Flow Condition – in order to guarantee the real–time flow of a telephone conversation between two parties, every voice–parcel is transferred across one net in one cycle. Thus, voice–parcels cannot be accumulated and are guaranteed to reach

their destinations in a **fixed** number of cycles.

9.4. The Voice Protocol

The voice protocol has three phases: (i) dial–up for constructing a bidirectional virtual path between the two parties; (ii) conversation, in which voice–parcels are transferred every cycle between the two parties; and (iii) termination of the conversation. The protocol is for 2D–R, but can be easily extended to other synchronous hypergraphs.

The telephones are interfacing the host bus, which resides outside of the network interface. The telephone interface may accommodate several telephones. The voice protocol is performed by the network interface, which receives and transfers speech voice–parcels from/to the telephone interface.

The protocol uses only primary routes, which utilize the least communication capacity of the net. On a 2D–R a phone conversation involves three nodes, as shown in Figure 9.1. The phone conversation is initiated via the SRC–node, the voice–parcels are sent on the SRC–net to the INT–node, and from there via the DST–net to the DST–node. The destination party is connected to the DST–node.

9.4.1. The dial–up phase

The first step, when a phone on a SRC–node calls a phone on a DST–node, is to determine the primary route, i.e., to select the INT–node. The objective of following steps is to reserve a virtual path from the SRC–node to the DST–node and back. The transfer over each part of this path (net) is multiplexed by several conversations. Reservation means that the port interface to the net allocates a free space for one

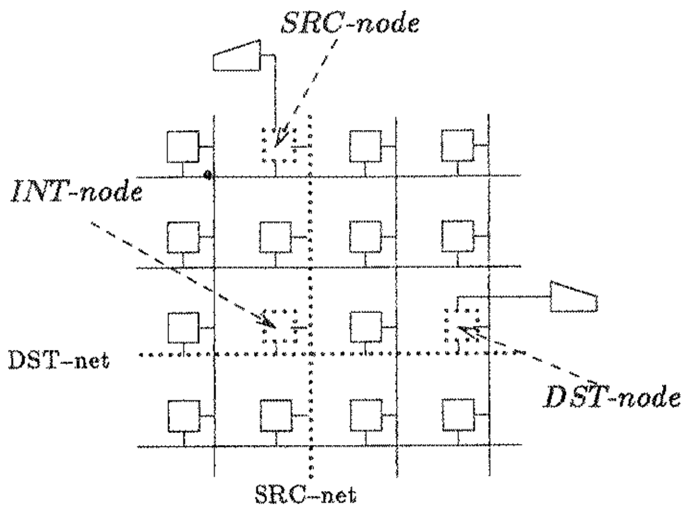


Figure 9.1: Integrating Voice on a 2D Regular Hypergraph

speech parcel in a DMS, which this port has reserved for its voice communication. If the port does not have a free space in a reserved DMS, it tries to reserve a free slot, and then it will allocate a free space for the parcel. If no free space is available, the dial–up procedure fails, and a "Line Busy" message is sent to the phone at the SRC–node.

Step 1 – the SRC–node tries to allocate a free space for a voice–parcel on the SRC–net. If the space allocation is successful, a dial–up message is sent to the INT–node, otherwise a "Line Busy" message is sent to the requester.

Step 2 – the INT–node tries to allocate a free space for two voice–parcels on the SRC–net and on the DST–net. If the space allocation is successful, a dial–up message is sent to the DST–node, otherwise a "Line Busy" message is sent to the requester via the SRC–node, and the SRC–node deallocates the free space already assigned.

Step 3 – the DST–node tries to allocate a free space for a parcel on the DST–net. If the allocation is successful, a "Line Ready" message is sent back to the SRC–node via the INT–node, otherwise a "Line Busy" message is sent back to the SRC–node via the INT–node, and both the INT–node and SRC–node deallocate the free spaces already assigned.

9.4.2. Conversation phase

For voice integration the system has two levels of synchronization (i) an event or a slot, and (ii) a cycle which starts whenever the value, modulo c, of the slot counter is one; i.e., the cycles of all the nodes are in phase. As a result an end–to–end voice synchronization is maintained, and the speech parcels which arrive at their destination can be converted back into voice in a constant rate (with maximum error of only $\pm 0.5 T_s$).

Each source and destination telephone generates one parcel per cycle. Since the dial–up procedure guarantees that there is a space for each parcel during each cycle, a voice–parcel is transferred from SRC–node to DST–node in two cycles. The INT–node will not transfer a voice–parcel to the DST–node in the same cycle it has been received; therefore, the parcels ordering at the DST–node are preserved.

The transmission of voice, during the conversation phase, is via the virtual multiplexing/demultiplexing mechanism, performed in two steps:

Multiplexing – a node broadcasts a packet of data during the DMS with several speech parcels, each with different destination on the net.

Demultiplexing – each node on the net receives a data packet and extracts the parcels with its destination and discards the rest.

The INT–node extracts parcels from the traffic on the SRC–net with a destination to the DST–node and transfers them to its orthogonal port which sends them to the DST–node via the DST–net.

9.4.3. Termination phase

The termination procedure frees the space allocated for the conversation during the dial–up phase. The SRC–node sends a "Terminate" message to the INT–node and then to the DST–node.

9.5. Analysis

Analysis evaluates the consequences of the voice integration on a 2D–R synchronous hypergraph. The analysis uses the parameters previously defined.

Example:–

For the discussion some numerical results are presented which use the following parameters:

- The speech is digitized by a sample of 8 bits every 125 μseconds (the bandwidth is 8,000 bytes per second)
- $T_p = 25{,}000$ μseconds (the duration of speech which is sampled into one voice–parcel of 1,600 bits)
- $BW_{net} = 10^9$ bits per second (the bandwidth of a net)
- $T_{DMS} = 32$ μseconds (the duration of one data minislot)
- $T_{PAR} = \frac{T_p}{125}\,\frac{8}{BW_{net}} = \frac{25{,}000}{125}\,\frac{8}{10^3} = 1.6$ μseconds (the duration of one parcel)
- $p = \frac{T_{DMS}}{T_{PAR}} = 20$ parcels in one data packet (during one DMS)

- $c = \dfrac{T_p}{T_s} \approx 700$ slots per cycle
- $n = 100$ nodes per net

9.5.1. End–to–end delay

The end–to–end delay is measured in cycles (T_p) and is defined as the time interval between the analog/digital conversion (ADC) and the digital/analog conversion (DAC). It takes one cycle for ADC of one parcel; at the end of the cycle the parcel is at the SRC–node. During the next cycle the parcel arrives at the INT–node and at the DST–node during the third cycle. The DAC, for regenerating the voice–parcel, can then start. Thus, it takes three cycles for a voice–parcel to get from the source phone to the destination phone; e.g., for $T_p = 25$ milliseconds the end–to–end delay is 75 milliseconds.

The time for internal data transfers within the interface is ignored since it is assumed to be much smaller than the cycle time. Also, the frame delay is ignored since it is only a few slots much less than the 700 slots of one cycle.

9.5.2. Reservation inefficiency

Reservation inefficiency is defined as the communication capacity which is wasted because of the voice reservation mechanism. Assume that the average utilization of a reserved slot is 0.5, so every port on a net might waste an average 0.5 DMS. Hence, the reservation inefficiency (ξ) is

$$\xi_{RESERV} = 0.5\frac{n}{c}100(\%).$$

For the above example it is $\xi_{RESERV} = 0.5\frac{100}{700}100 = 7\%$.

Note: (i) The wasted space is actually free and can be used by the node for data transfers or for dial–up and termination messages. (ii) The reservation inefficiency decreases as the number of nodes decreases. (iii) The number of slots in one cycle (c) increases linearly as the BW_{net} increases; therefore, in a higher communication bandwidth the reservation inefficiency decreases (assuming that all other parameters are not changed). This last result is especially important since it indicates a direct advantage for using channels with a higher bandwidth.

9.6. Routing by Local Balancing

The routing algorithm is performed by the SRC–node before beginning the dial–up protocol. In general, telephony communication is easier to balance, since each conversation duration is about four orders of magnitude longer than the time slot. Therefore, load balancing can be viewed as a static procedure rather than a dynamic one.

The proposed routing algorithm has the following principles:

(i) each node has a reservation register at each port,

(ii) each node does not reserve more than it actually needs,

(iii) only primary routes are considered,

(iv) reservation criterion – the selection between the two primary routes is made such that the total number of reserved slots on the two orthogonal nets is the same, i.e., the two reservation registers have reserved the same number of slots.

The major advantage of this procedure is that it can be done in a completely distributed manner and without exchanging any additional information. If the nets systems are more or less uniform, this algorithm may have an efficient performance.

9.7. Discussion

The integration of voice telephony into the synchronous optical hypergraph has been shown to be straightforward. From the hardware point of view there are two additional requirements: (i) reservation register of c cells at each port, and (ii) comparator for the virtual demultiplexing.

The low–dimension, high–bandwidth hypergraph has two major advantages over higher dimension hypergraph (or point–to–point network):

(1) The end–to–end delay of the voice–parcel is proportional to the length of the path, which is measured by the number of nets in the path. Therefore, a lower dimension with fewer nets in its path results in a shorter end–to–end delay.

(2) The inefficiency for the integration of voice is smaller as the bandwidth gets higher, as shown in the following equations:

$$\xi_{RESERV} = 0.5\frac{n}{c}100(\%).$$

Since, (i) $c = \frac{T_p}{T_s}$, (ii) r is the size of a slot in bits, and (iii) $T_s = \frac{r}{BW_{net}}$, the ξ_{RESERV} is

$$\xi_{RESERV} = 0.5\frac{n}{T_p}\,\frac{r}{BW_{net}}100(\%).$$

CHAPTER 10.

DISCUSSION AND CONCLUSIONS

10.1. General

The concept of a synchronous optical hypergraph has potential for many applications, of which the metropolitan area network is only one. The hypergraph architecture may be extended in size by combining satellite networks, or it may be implemented, in a very small area, as a machine for processing data rather than as a system for transferring data. Another possible use of the synchronous hypergraph is in real–time control of industrial automation. In general, almost any real–time system which is spread over a large area can be realized by the synchronous hypergraph.

The two most important attributes of this system are

(1) The ability to transfer efficiently small and large messages at high bandwidth by using the conservative code. Thus, it is possible to exchange control messages efficiently, which enables the integration of various network functions in a uniform manner.

(2) Efficient global synchronization and total event ordering, enables the straightforward integration of voice and simplifies the design of distributed parallel algorithms.

The core of this system is a centralized, passive, optical star coupler, which is very reliable, and which splits the energy among the receiving nodes in an optimal way: if there are n nodes, each node receives $\frac{1}{n}$ of the energy. The links connecting the nodes to the centralized switch are single-mode, very low-loss, optical fibers. Thus, longer links do not impose any significant reduction to the energy received by each node, and it is better to let the signal travel a longer distance and to split equally the signal energy, rather than have shorter links but without an equal division of the energy, as in the cases of linear bus and passive ring.

10.2. Thesis Discussion

The following discussion shows that the thesis of this dissertation "*A low-dimension, high width (the number of ports on a net), globally-synchronized, optical hypergraph, can be efficiently (with low overhead) managed and controlled in a distributed manner,*" is an immediate result of the design and analysis of the system, as presented in the previous chapters. This design is not arbitrary, but is rather a logical outcome of the technological advances in optical communication and semiconductors. Furthermore, the design gives specific solutions to problems in the areas of communication and computation.

The argument is illustrated in Figure 10.1. The different attributes of the optical hypergraph are circularly related to one another. In the discussion only the major relevant arguments are mentioned.

The immediate consequence of recent technological advances is that communication bandwidths about 100 times greater than at present are feasible. Therefore, a medium with such high bandwidth can support multiple nodes. Note that as the bandwidth increases the width (number of nodes) of the net can be increased as well. The high

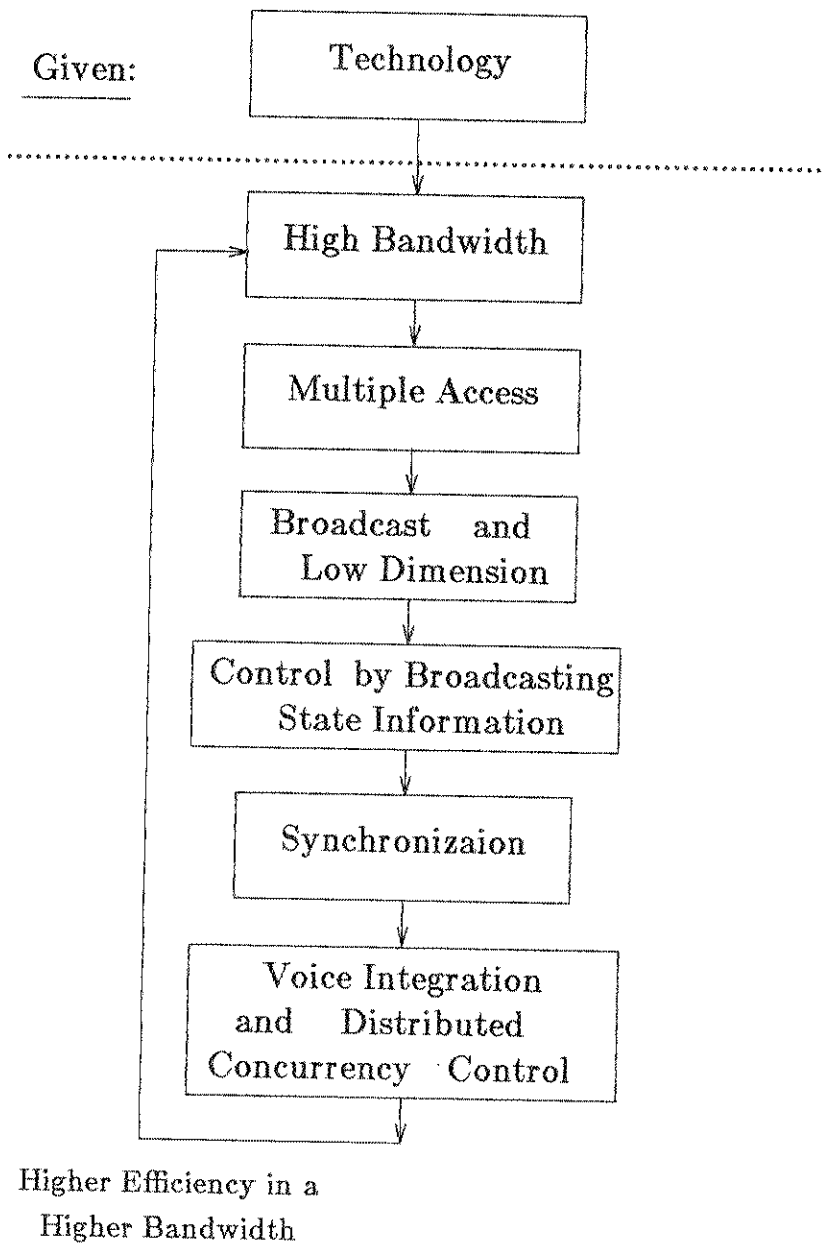

Given:
Technology
High Bandwidth
Multiple Access
Broadcast and Low Dimension
Control by Broadcasting State Information
Synchronizaion
Voice Integration and Distributed Concurrency Control
Higher Efficiency in a Higher Bandwidth

Figure 10.1: Thesis Discussion

bandwidth is well suited for the realization of the conservative code, which enables nodes to efficiently transmit and receive very short messages for the control of the multiple-access medium.

For a given total number of nodes, p, the dimensionality, D, of the network is related to its width, W, by

$$p = W^D \quad or \quad D = \log_W p$$

The broadcast nature of the transmission, together with a low dimensionality hypergraph, enable state information to propagate rapidly through the network, which is essential for the periodic exchange of state information and the implementation of global synchronization, so as to maintain total event ordering. Using this basic construction, the efficient integration of voice becomes very simple. The efficiency of voice integration increases with bandwidth. With this argument, the loop is closed. The period of exchanging state information gets shorter as the bandwidth gets higher. Thus, distributed algorithms are more efficient, since they use more recent state information.

10.3. On the Virtue of an Optical Hypergraph

The design principles of this system are applicable to different topologies; the design of 2D–R and 2D–P can be easily extended to an arbitrary hypergraph. This is important because of

(1) Graceful degradation – in principle, any hypergraph can degrade its operation to another arbitrary hypergraph, which is smaller in size. If a node or a port ceases to exist, only its neighbors on its connected nets need be aware of this and make sure not to route any message to it. They may use an alternative route or send a

message to the originating node.

(2) Graceful upgrading – the opposite of the previous property. In many practical cases the system is not implemented at once, either because the requirements are not known or due to lack of resources. The optical hypergraph can be extended gradually, the basic configuration being a single net, and more nets can then be added later as needed.

10.4. Synchronous and Asynchronous Computations

The system is synchronous in its macrolevel (slot duration) and is asynchronous in its microlevel. The computations that are distributedly performed on this system can be viewed as asynchronous computations with bounded time, where the bound is the time slot or frame. It is interesting to note that if the slot size is determined by a typical page or file in the system (e.g., 4K bytes), then the bound on asynchronous computation is inversely proportional to the baud rate. This is another reason why increasingly higher bandwidth is significant.

For example, with a slot length of 8k bytes and a baud rate of 1 gigabit/second, the slot duration is 64 microseconds, which is shorter than the time of most software processes, even for fast, general purpose machines. Thus, if the basic computational step is not smaller than the bound asynchronous computation, then it is reasonable to consider that the distributed execution is synchronous.

10.5. Reduction of Software Complexity

One of the major design objectives is to reduce the system's software complexity. In the optical hypergraph design this objective has been achieved by several means:

(1) Open mode concurrency control, which operates without an explicit acknowledgement. This alleviates the necessity for special acknowledgement messages. Moreover, the open–mode concurrency control requests are performed by the interface, independent of the distributed operating system software.

(2) Total event ordering, simplifies the execution of distributed algorithms and the scheduling mechanism of distributed operating systems. With total event ordering it is also easier to maintain the bookkeeping of events in the system. The temporal order is the natural order in any system. Thus, with total event ordering it is simpler to ensure the serializability of a distributed procedure.

(3) Implementing "traditional" operating system functions (like routing or buffer management) by the interface in real time, by a dedicated hardware or firmware.

10.6. Further Considerations

The principles for constructing the synchronous hypergraph can be extended in two ways:

(1) Higher dimensionality, such as 3D regular hypergraph, with three ports per node. The network then can have 100,000 to 1,000,000 nodes. Increasing the dimensionality is done only when it is not possible to accommodate all the nodes on a lower–dimension hypergraph.

(2) Satellite network, another interesting extension, can be done by combining several optical hypergraphs via satellite networks. The satellite network has an **active star topology**, so that synchronization among its nodes should be as simple as for an optical star. Thus, coherent integration with the optical nets is possible but not simple. The major difference is that the satellite network exhibits much longer delays (about 1000 times).

REFERENCES

[Abra77] N. Abramson, "The Throughput of Packet Broadcasting Channels," *IEEE Trans. on Comm.*, Vol. COM–25, No. 1, January 1977 (pp. 117–128).

[Aviz78] A. Avizienis, "Fault–Tolerance: The Survival Attribute of Digital Systems," *Proceedings of the IEEE*, Vol. 66, No. 10, October 1978.

[Awer85] B. Awerbuch, "Complexity of Network Synchronization," *J. of ACM*, Vol. 32, No. 4, October1985 (pp. 804–823).

[Berg73] C. Berge, *Graphs and Hypergraphs.* Amsterdam: North–Holland Publishing Company, 1973.

[FiTo86] M. Fine and F. A. Tobagi, "Packet Voice on a Local Area Network with Round Robin Service," *IEEE Trans. on Comm.*, Vol. COM–34, No. 9, September 1986 (pp. 906–915).

[Fran70] P. A. Franaszek, "Sequence–state Method for Run–length–limited Coding," *IBM J. of Res. and Develop.*, July 1970 (pp. 376–383).

[Fran82] P. A. Franaszek, "Construction of Bounded Delay Codes for Discrete Noiseless Channel," *IBM J. of Res. and Develop.*, July 1982 (pp. 506–514).

[GoWo85] P. M. Gopal and J. W. Wong, "Analysis of a Hybrid Token–CSMA/CD Protocols for Bus Networks," *Computer Networks and ISDN Systems 9*, 1985 (pp. 131–141).

[Grub81] J. G. Gruber, "Delay Related Issues in Integrated Voice and Data Networks," *IEEE Trans. on Comm.*, Vol. COM–29, No. 6, June 1981 (pp. 786–800).

[HoOs75] S. L. Hong and D. L. Ostapko, "Codes for Self–clocking, AC–coupled Transmission: Aspects of Synthesis and Analysis," *IBM J. of Res. and Develop.*, July 1975 (pp. 358–365).

[JoIy84] S. Joshi and V. Iyer, "New Standards for Local Networks Push Upper Limits for Lightwave Data," *Data Communications*, July 1984 (pp. 127–138).

[Kama87] Ahmed E. Kamal, "Star Local Area Network: A Performance Study," *IEEE Trans. on Computers*, Vol. C–36, No. 4, April 1987 (pp. 483–499).

[Kapr85] F. P. Kapron, "Fiber Optic System Tradeoffs," *IEEE Spectrum*, March 1985 (pp. 68–75).

[Knut86] D. Knuth, "Efficient Balanced Codes," *IEEE Trans. on Info. Theory*, January 1986.

[KuRo81] H. T. Kung and J. T. Robinson, "On Optimistic Methods for Concurrency Control," *ACM Trans. on Database Systems*, Vol. 6, No. 2, June 1981 (pp.

213–226).

[Lacr84] J. Lacroix, "Line Signals in Submarine Digital Telephone Links Using Optical Fibers," *J. of Lightwave Technology* (IEEE), December 1984.

[Lamp78] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Comm. of the ACM*, Vol. 21, No. 7, July 1978 (pp. 558–565).

[Lazar85] A. A. Lazar, A. Patir, T. Takahashi, and M. El Zarki, "MAGNET: Columbia's Integrated Network Testbed," *IEEE J. on Selected Areas in Comm.*, Vol. SAC–3, No. 6, November 1985 (pp. 859–871).

[LeBo83] E. S. Lee and P. I. P. Boulton, "The Principles and Performance of Hubnet: A 50 megabit/Sec Glass Fiber Local Area Network," *IEEE J. on Selected Areas in Comm.*, Vol. SAC–1, No. 5, November 1983.

[LeBoIk84] Lee, Boulton and Ikeman, "A Glass–Fiber Rooted–Tree Local Area Network," *FOC/LAN'84.*

[Leis84] E. L. Leiss, "Data Integrity in Optical Disks," *IEEE Trans. on Comp.*, Vol. C–33, No. 9, September 1984 (pp. 818–827).

[LMHo86] Ming–Kang Liu, David G. Messerschmitt and David A. Hodges, "An Approach to Fiber Optics DATA/VOICE/VIDEO LAN," *INFOCOM 1986.*

[Mang83] G. R. Mangus, "LANFOTS – A Fiber Optic Transmission System for A Local Area Network," *FOC/LAN'83.*

[Marh84] M. E. Marhic, "Combinatorial Star Couplers for Single–Mode Optical Fibers," *FOC/LAN'84.*

[McOf87] P. McKinley and Y. Ofek, "Resource Sharing in a Synchronous Optical Hypergraph," To be presented in the Symposium on the *Simulation of Computer Networks.*

[Mont83] W. A. Montgomery, "Techniques for Packet Voice Synchronization," *IEEE J. on Selected Areas in Comm.*, Vol. SAC–1, No. 6, December 1983 (pp. 1022–1028).

[NTM85] M. M. Nassehi, F. A. Tobagi and M. E. Marhic, "Topological Design of Fiber Optics Local Area Networks with Application to Expressnet," *SEL Technical Report No. 85–271*, Stanford University March 1985.

[Ofek87a] Y. Ofek, "A Family of Conservative Codes with Block Delimiters for Decoding without a Phase–Locked Loop," *The 1987 ACM Computer Science Conference.*

[Ofek87b] Y. Ofek, "An Encoder/Decoder System and Methodology Utilizing Conservative Coding with Block Delimiters, for Serial Communication," Patent pending (U.S.).

[OfFa87a] Y. Ofek and M. Faiman, "A Digital Interface to a One Gigabit/sec Multiple–Access Fiber–Optic Network," *The Sixth Annual IEEE Phoenix*

Conference on Computers and Communications (1987).

[OfFa87b] Y. Ofek and M. Faiman, "Distributed Global Event Synchronization in a Fiber Optic Hypergraph Network," To be presented in the *7th International Conference on Distributed Computing Systems.*

[Pers83] D. P. Personick, "Review of Fundamentals of Optical Fiber Systems," *IEEE J. on Selected Areas in Comm.*, Vol. SAC–1, No. 3, April 1983.

[RePa85] S. Renben and P. C. Patton, "BCA: A Bus Connected Architecture," *The 1985 International Conference on Parallel Processing (ICPP 85).*

[SRNJB83] Schmidt, Rawson, Norton, Jackson and Baily, "Fibernet II: A Fiber Optic Ethernet," *IEEE J. on Selected Areas in Comm.*, Vol. SAC–1, No. 5, November 1983.

[Seve80] R. H. Severt, "Encoding Schemes Support High Density Digital Data Recording," *Computer Design*, May 1980 (pp. 181–190).

[Sore84] H. Sorensen, "Use of Standard Modulation Codes for Fiber Optic Link Optimization," *FOC/LAN'84.*

[Sze85] D. T. W. Sze, "A Metropolitan Area Network," *IEEE J. on Selected Areas in Comm.*, Vol. SAC–3, No. 6, November 1985 (pp. 815–824).

[Tane81] A. S. Tanenbaum, *Computer Networks.* Prentice–Hall, Inc. 1981.

[TCJ83] C. Tseng, B. Chen, and A. Jalail, "Implementation of D–NET at TRW Technology Research Center," *FOC/LAN'83.*

[Ullm84] J. D. Ullman, "Flux, Sorting, and Supercomputer Organization for AI Applications," *J. of Parallel and Distributed Computing 1*, 1984 (pp. 133–151).

[VlLe84] D. Vlack and H. R. Lehman, "An Experimental Digital Video Switching Architecture," *ISS'84 Florence.*

[WiFr83] A. Widmer and P. Franaszek, "A DC–Balanced, Partitioned–Block, 8B/10B Transmission Code," *IBM J. of Res. and Develop.*, September 1983.

[Witt81] L. D. Wittie, "Communication Structure for Lare Network of Microprocessors," *IEEE Trans. on Comp.* Vol. c–30, No. 4, April 1981.

VITA

Yoram Ofek was born in Kibbutz Ramat David, Israel. In 1979, he completed his B.S. in Electrical Engineering at the Technion, the Israel Institute of Technology. After graduation, he worked with the Israeli Armament Development Authority and then at Fermi National Accelerator Laboratory. He started his graduate studies in the Department of Electrical and Computer Engineering, at the University of Illinois, Urbana–Champaign, in January 1984. He received his M.S. degree in May of 1985, and his Ph.D. degree in May of 1987. Yoram is married to Barbara, and they have three children: Tidhar, Gidon, and Daphna.