National Library of Canada
Acquisitions and Bibliographic Services Branch
395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale du Canada
Direction des acquisitions et des services bibliographiques
395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Canada

Optical MANs Based on the de Bruijn Graph

by

Zhi Feng, B.Eng.

A thesis submitted to the
School of Graduate Studies and Research
in partial fulfillment of the requirement for the degree of

Master of Applied Science

Ottawa-Carleton Institute for Electrical Engineering

Department of Electrical Engineering
Faculty of Engineering
University of Ottawa

June 1994

ISBN 0-612-00531-3

Canada

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

Abstract

This thesis proposes and studies multihop lightwave networks based on the de Bruijn Graph(dBG) and their hierarchical structures. Routing algorithms that take advantage of the inherent structure of the network are studied. Three new algorithms that would progressively improve on the mean path length performance are proposed, and the network performances are studied in terms of mean path length, network throughput, delay, locality factor and the propagation delay. The study shows that the performance of this type of network topology is desirable and is comparable to other types of multihop systems. It may be considered as a candidate for optical Metropolitan Area Networks. The optical implementation and design criteria are also discussed in the thesis. It shows that such networks are feasible with today's lightwave technology and can also be easily adapted to the more advanced technique in the future.

ACKNOWLEDGMENTS

I am deeply grateful to Dr. Oliver W.W. Yang, my supervisor. for his consistent knowledgeable guidance, extremely helpful comments and suggestions, as well as teaching me the productive scientific approach during the whole process of this work. I would also like to express my appreciation to the members of the photonic research group in University of Ottawa by whom I can always learn the advanced optical technologies. I am thankful to my colleagues in our computer communication research group for their fruitful discussions and always being available to help.

Lastly, I owe a great debt of thanks to my family, especially my wife, Wei-dong, for her understanding, encouragement and endless support without which this work may never have been undertaken.

This research has been supported by a TRIO Research Assistantship under the Thrust of Photonic Networks and Systems and an NSERC Operating Grant under contract #OGP0042878.

Table of Contents

List of Figures

List of Tables

Glossary of Notations

α:	Locality factor.
C:	Link capacity.
Δ:	Degree of network.
D:	Diameter of dBG network.
i, j, k:	Integer constants.
J:	Number of links in network.
Λ:	Number of wavelengths.
λ:	Poisson arrival rate.
L_i:	Edge load on link i.
$1/\mu$:	Mean packet length in bits.
m:	Number of clusters.
N:	Number of nodes in network.
p:	degree of ShuffleNet.
r:	Propagation time ratio.
τ:	Propagation time.
$\overline{H}$ (node-i):	Mean path length of node i.
$\overline{H}$:	Mean path length of network.
$\overline{B}$:	Mean path length of bridge node.
$\overline{D}$:	Average packet delay.
(c,r):	Node's column and row coordinate in ShuffleNet.

Chapter 1
Introduction

1.1 Fiber Optic Networks

Reviewing the computer networks in the past, one can identify three generations of technology employed in the physical transmission of information[GREE91, MUKH92a].

First-generation networks are those built before the emergence of fiber optic technology. They are based on copper-wire or microwave-radio technology including Ethernet, IEEE 802.4 token bus, IEEE 802.5 token ring, etc..

Second-generation networks employ fibers in traditional architecture. One example is the upgrade of long-haul trunks in a wide area networks(WAN) from copper or microwave-radio to fiber connections. It exploits the very low attenuation or transmission loss of optical fibers in comparison with the best copper conductors. Fibers that have been fabricated have losses as low as 0.15-0.2dB/km; there is also ongoing laboratory research on exotic materials that could have attenuation less than 0.01dB/km[GREE93]. If these ever become practical, one could be able to look into one fiber end on one side of the Atlantic and see someone blinking a flashlight into the other end on the other shore.

Other examples of second-generation networks include new designs such as the Fiber Distributed Data Interface(FDDI) ring network, IEEE 802.6 distributed queue dual bus(DQDB) for local area network(LAN) and metropolitan area network(MAN) environments and Broadband Integrated Services Digital Network(BISDN) for WANs. Although some improved performance can be achieved, the limitation of the second-generation networks is due to the electronic front-ends employed at the networks' nodes. The

information-carrying capacity of optical fiber is nearly four orders of magnitude greater than peak electronic processing speeds[MUKH92a].

In third-generation networks, fiber is used because of its unique properties, referring to the vast bandwidth which is approximately 25 to 30THz corresponding to the two low-loss regions in a single-mode fiber [GREE93]. These networks employ totally new approaches, such as the various access methods introduced in the next section, to exploit the unique properties of fibers. The third-generation networks, therefore, are able to meet the needs of emerging high-bandwidth applications, such as "fiber to the office", "fiber to the home" and "information super highway" services providing multiple connections of high-definition television(HDTV), digital audio, and other new applications. Specifically, the third-generation networks are "all optical" in nature in the sense that once the information enters into the network, it will remain in optical domain until it reaches the destination. There is no electronic processing except for the overhead of the information packets. In the near future, purely optical networks will be possible as the fast developing trend of optical technologies continues.

The focus of this thesis is on the third-generation all optical networks.

1.2 Access methods of Fiber Optic Networks

Due to the bottleneck at the electro-optical interface, the maximum rate at which each end user can access the network is limited by the electronic speed. One key approach to exploit the enormous optical bandwidth is to use the network architectures and protocols such that they allow the concurrence of packet transmission among multiple-users. However, these different transmissions must be multiplexed on the fibers so that they do not interfere with one another. Generally there are three different ways to do this today: wavelength-division multiplexing(WDM), time-division multiplexing(TDM), and code-division multiplexing(CDM). Of these three, the most practical alternative today for networks is WDM[GREE93, SIVA92].

In WDM networks, the different transmissions concurrently supported on the optical fiber are at different optical wavelengths or frequencies. One common configuration is shown in Figure 1.1. In this example, there are N=4 nodes(users) connected by a broadcast star coupler[SUDH91], and each station transmits packets using different wavelengths according to the packet destination node. If enough wavelengths are available and a proper control protocol is employed, there will be no conflict in the transmissions of information packets, and any two arbitrary nodes can directly communicate optically without interfere with each other.

In such WDM networks, if the number of wavelengths is large enough, the huge bandwidth of the optical fiber can be tapped effectively. It is the purpose of this thesis to study the performance of the WDM lightwave networks.

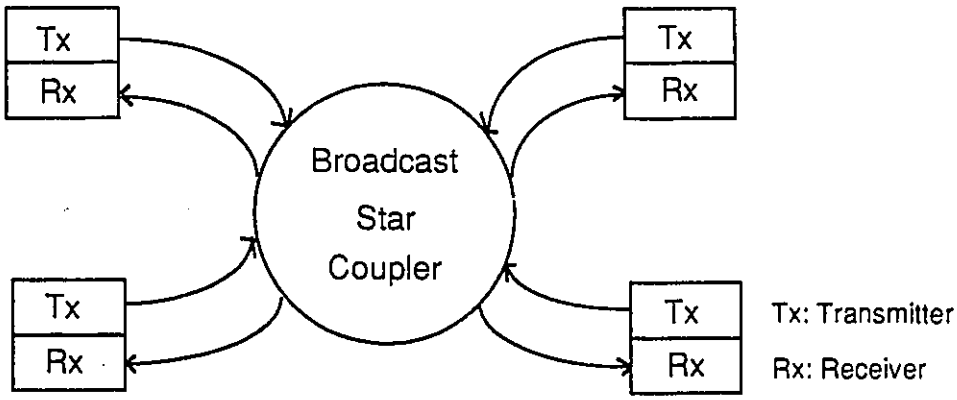


Fig.1.1: A star topology for WDM networks.

1.3 Architectures of WDM Networks

In general, architectures for WDM fiber optic networks fall under two main categories: single-hop systems and multihop systems[MUKH92a, MUKH92b].

1.3.1 Single-hop Systems

In single-hop systems, each node is equipped with fixed or/and tunable optical transceiver(s) accessing to a number of wavelengths, say Λ. All nodes in the network share these Λ wavelengths using a multiple access protocol. An information packet can reach its destination node by only one hop. Common topologies of single-hop systems include the passive star[BRAC90, DOWD91] and the bus[CHLA88].

Note that in order to receive successive transmissions from different nodes, the receiver must tune quickly from one wavelength to another. The tuning speed of commercially available tunable transceivers today is of the order of milliseconds[SIVA92] while the packet transmission time in a gigabit network last no more than a few microseconds. Consequently, such networks are suitable for circuit-switched, not packet-switched, operation. In order for such WDM networks to be packet-switched, two things are required: rapidly tunable optical transceivers(in a small fraction of a packet transmission time) and an efficient and easily implemented multiple-access protocol. A lot of effort, summarized in [MUKH91a], has been directed on both problems but much progress needs to be made before such networks become practical, especially for large networks[SIVA92].

1.3.2 Multihop Systems

Unlike the single-hop systems, packet-switched lightwave networks based on multihop logical topologies can be implemented with current technology.

A.S.Acampora[ACAM87] first suggested multichannel multihop technique for utilizing the vast bandwidth of the fiber optic media using fixed wavelength optical transceivers. Since there is no direct optical path connecting every node-pair in the network, packets may have to go through more than one hop before they reach the destination node. However, the requirement of optical transceivers in these networks is much lower than single-hop systems, and no multiple-access protocols are necessary. An experimental

multihop network that is currently operational and in the process of being expanded is Columbia University's TeraNet[GIDR91].

Some popular topologies of multihop systems include ShuffleNets, Manhattan Street Networks, Hypercube, and de Bruijn Graph. A brief review on these topologies is introduced in the next section.

1.4 Common Topologies of Multihop Systems

1.4.1 ShuffleNets

ShuffleNets were first proposed for multichannel multihop lightwave application by A.S.Acampora in 1987 [ACAM87]. In general, a (p,k) ShuffleNet consists of $N=kp^k$ (k=1,2,... & p=1,2,...) nodes arranged in k columns of p^k nodes each. Each node in the k-th column is connected to the first so that the interconnections graph can be visualized as wrapped around a cylinder. An example of (2,2) ShuffleNet is shown in Figure 1.2.

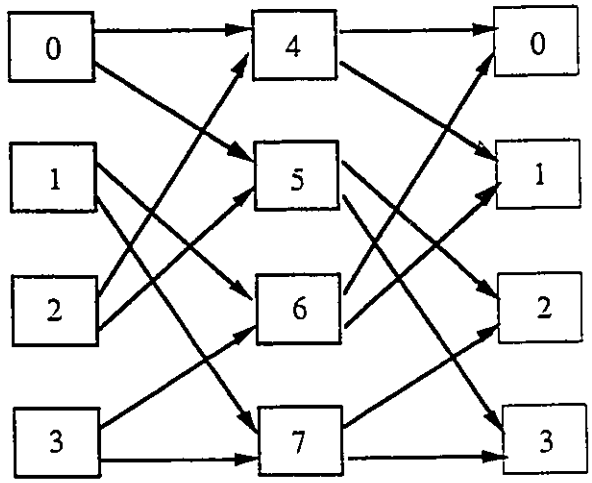


Fig. 1.2: A (2, 2) ShuffleNet.

Because there are multiple minimum-hop routing paths for some source-destination pairs, there are many possible shortest path routing algorithms for ShuffleNet. One simple scheme is outlined in [ACAM89]. A node in a (p,k) ShuffleNet is assigned the

address (c,r) where $c \in \{0,1,\ldots,k-1\}$ is the node's column coordinate(labeled 0 to k-1, left to right) and $r \in \{0,1,\ldots,p^k-1\}$ is the node's row coordinate(labeled o to p^k-1, form top to bottom, using base-p digits). Thus one may write $r=r_{k-1}r_{k-2}\ldots r_2r_1r_0$. With this addressing scheme, the following routing decisions are made when an arbitrary node $(\hat{c},\hat{r})$ receives a packet with destination address $(c^d,r^d) \neq (\hat{c},\hat{r})$. Let X denote the number of columns between the current location $(\hat{c},\hat{r})$ and the destination (c^d,r^d):

$$X = \begin{cases} (k + c^d - \hat{c}) \bmod k & if\ c^d \neq \hat{c} \\ k & if\ c^d = \hat{c} \end{cases}$$

Then the node uses the p-ary digit r (which is part of the destination address in the packet header) to determine which of its p output ports to use for routing the packet to (c^d,r^d); specifically, the packet is routed to the node with identity

$$((\hat{c}+1)\ mod\ k, \hat{r}_{k-2}\hat{r}_{k-3}\ldots\hat{r}_2\hat{r}_1\hat{r}_0\hat{r}^d_{X-1}).$$

In addition, a number of papers have appeared in the literature dealing with the routing problems in ShuffleNet[ACAM91, KARO91, KRIS90, ZHAN90, ZHAN91].

According to the shortest path routing algorithms, the mean path length of the ShuffleNet can be obtained as

$$\overline{H} = \frac{kp^k(p-1)(3k-1) - 2k(p^k-1)}{2(p-1)(kp^k-1)}$$

ShuffleNets perform well especially when the diameter of the network, defined as the number of maximum hops between any two nodes, is small (when the number of nodes is relatively small)[SIVA91].

Bidirectional ShuffleNet is also proposed in the literature, such as in [AYAD93]. Instead of only connecting each node in the networks to p different nodes in the next column, the Bidirectional ShuffleNet lets each node to be connected to a "mirror" image of nodes from the previous column; It has been shown that such kind of networks can achieve higher efficiency compared to the conventional ShuffleNet.

1.4.2 Manhattan Street Networks(MSN)

The Manhattan Street Networks were first proposed by Maxemchuk[MAXE85]. The nodes in a MSN are connected in a grid where the traffic in adjacent rows and columns moves in the opposite directions. The wrap-around links in the network have the effect of logically placing the grid on the surface of a torus so that there are no corners.

The regular MSNs have even number of rows and columns as shown in Figure 1.3. Although the MSNs are highly modular and easily growable[MAXE85, MAXE87], they have a limit on the node degree which is 2 or 4 in the bidirectional networks, and they have larger mean path length compared to ShuffleNets[MUKH92b].

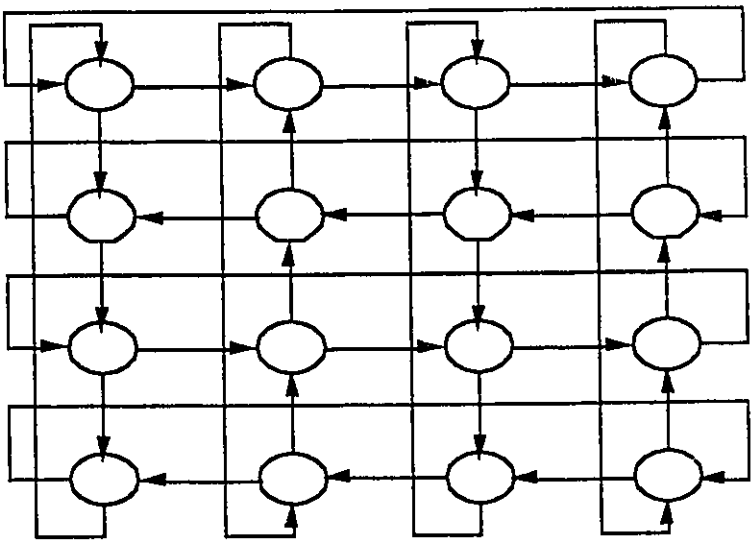


Fig. 1.3: A 4 x 4 Manhattan Street Network.

The deflection routing method can be used effectively in the MSNs. To avoid the electro-optical conversion and buffering, the intermediate node receiving a packet may deflect or intentionally misroute the packet which has just lost its contention to the free outgoing link with the hope that the packet will eventually find its way back to its destination. The packet may go through a longer path but avoid the congested part or hot spots in the

network. The advantage of the MSNs is that if a packet is forced to take the wrong path, the increase in path length to the destination is never more that four.

The addressing scheme and the routing algorithms are discussed in detail in [MAXE87]. Several distributed routing rules are investigated. Simple routing rules that use the regular structure of the network are compared to shortest path algorithms and random routing strategies.

1.4.3 Hypercube

The simplest form of the hypercube interconnection pattern is the binary hypercube[LIB92]. A p-dimension binary hypercube has $N=2^p$ nodes, each of which have p neighbors. Any node, say i, with an arbitrary binary address will have as its neighbors those nodes whose binary address differ form node i's address in exactly one bit position. An 8-node binary hypercube is shown in Figure 1.4.

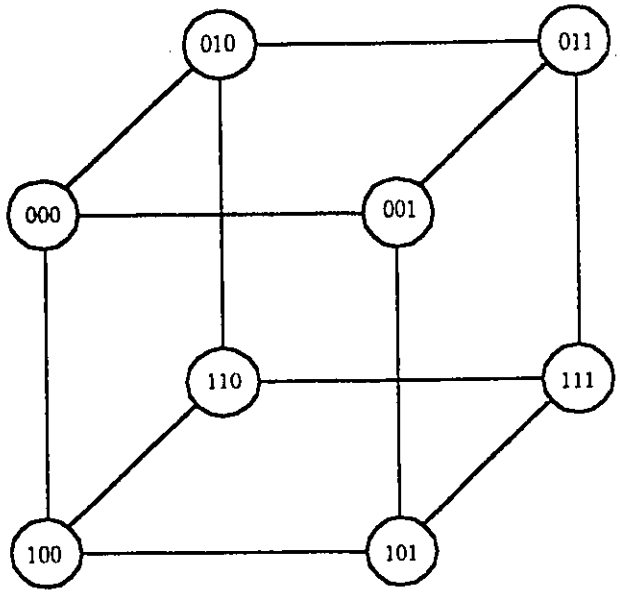


Fig.1.4: An 8-node binary hypercube.

Generalizing the nodal address notations in binary hypercube results in the generalized hypercube structure[LIB92]. Such structures can be more flexible in accommodating different number of nodes and their interconnection patterns.

Routing in binary hypercubes can be simple. Because any node-i with an arbitrary binary address will have as its neighbors those nodes whose binary addresses differ from nodes i's address in exactly on bit position, information packets can be sent through the node whose address is matching one more bit with the destination node's address. Routing in general hypercubes is in similar fashion.

The merits of the hypercube structure are its small diameter($\log_2 N$ in binary hypercube) and short mean path length($(N\log_2 N)/(2(N-1))$ in binary hypercube). However, its major disadvantage is that the nodal degree increases logarithmically with N. It means that in a larger network, the implementation of the node is more complex.

The hierarchical structures of the hypercube networks have been studied in [DAND90, DAND91]. It has been shown that the hierarchical networks provide better cost-benefit ratios than the corresponding non-hierarchical interconnection networks.

1.4.4 De Bruijn Graph(dBG)

Since the construction of this type of networks is the subject of this thesis, details will be given in Chapter 2. Instead, an example of (2,3) dBG which has N=8 nodes is shown in Figure 1.5.

Compared with ShuffleNets, the dBGs seem to be more efficient in term of the number of nodes accommodated[GREE93]. Moreover, it is found that, in general, for a given maximum nodal in- and out-degree and mean path length between nodes, networks based on the dBGs can support a much larger number of nodes than can ShuffleNets[SIVA91].

This thesis will study the performance of dBGs and their hierarchical structures as well. Therefore, a brief literature review on the dBGs is given in the following section in order to introduce the background and the context of the thesis.

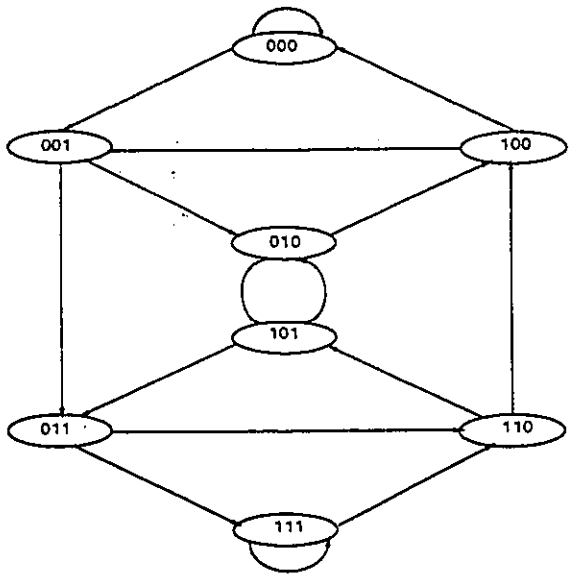


Fig. 1.5: A (2,3) dBG network.

1.4.5 Physical Versus Logical Topologies

The physical topology depicts the actual underlying layout of the network; i.e., how the nodes are connected by a physical transmission medium. It can be a broadcast type or non-broadcast type, a star, bus or a tree, or even the same as the logical topology. By the logical topology, one can describe the dedicated connections between certain pairs of nodes. One can carefully select a logical topology to impose on a physical topology or design a physical topology based on the logical topology.

The topologies for multihop systems mentioned in Section 1.4 are generally referred to as logical topologies. The actual physical topology can be varied according to the network design.

The advantage of imposing a logical topology on a physical topology in multihop systems is to avoid tunable transmitters and receivers which are not really practical at the present stage. If the physical topology is a passive star, for example, a transmission from a node will reach all other stations. The logical topology of the network can be ShuffleNet, Manhattan Street Network or de Bruijn Graph by using sufficient optical channels at different wavelengths. Thus the network capacity can be significantly enhanced by the WDM technique. In such implementation, most transmissions would hop back and forth across the network many times before reaching their destinations.

Our analysis in this thesis in terms of mean path length of the network, packet delay and throughput is based on the traffic intensity at each logical hop. An actual physical link may contain several logical paths of different wavelengths. However, it may be worthwhile to point out that, from the network design of view, such kind of topology can even be used as the actual physical topology and our analysis results are still valid. In that case, there is not necessary to use WDM technique because all connections are strictly point-to-point. Thus we can even make use of yesterday's technology in building a lightwave network.

1.5 Metropolitan Area Networks(MAN)

The above discussed topologies are all suitable for constructing optical Metropolitan Area Networks that belong to a type of computer networks as opposed to Local Area Networks(LAN) and Wide Area Networks(WAN).

While LANs are typically used to interconnect computers and PCs within a relatively small geographic area, such as within an office, a building, or a small cluster of buildings (a campus), WANs are designed to interconnect computer systems and smaller networks over very large geographic scopes, from a city to a country to the entire planet. MANs, in many ways, fit between LANs and WANs and represent the first efforts to effectively bridge the gap between local and wide area network services and technologies. MANs are intended for the interconnection of LANs and hosts in a large campus or

citywide area, say 100km or more geographic scope[KESS91]. MANs can also accommodate much more nodes than LANs do. From the network speed point of view, MANs operate at speeds between 45 and 150 Mbps while LANs usually operate at speeds between 50 and 100Mbps. However, with optical technology nowadays, MANs can even operate at a speed approaching gigabits per second. That makes possible to realize the services such multimedia and real time communications.

To support the services offered by MANs, there are some trials and availability of products nowadays including SONET(Synchronized Optical Network), frame relay, SMDS(Switched Multi-megabit Data Service), and Broadband ISDN(Integrated Services Digital Network).

This thesis is intended to propose topologies based on the de Bruin Graph that are suitable for optical multihop systems by which optical MANs can also be constructed.

1.6 Literature Review On The De Bruijn Graphs

The application of de Bruijn graphs(dBG) was first found in the seventies when researchers studied the design of feedback shift registers[LEMP70].

The dBG was first proposed by D.K.Pradhan in 1982 as a communication architecture for distributed processors[PRAD82]. It was shown that this architecture enjoys the advantages of the low complexity, short routing distance and ease of diagnosis. Furthermore, it was also shown that it is fairly easy to tolerate faults in the network. Given a dBG as a function of two parameters, say Δ and D(to be defined later in the thesis), it was shown that the node-connectivity of this dBG is at least Δ. A distributed fault-diagnosis algorithm was presented which allows for diagnosis of faults without the use of a central observer.

Since then, the connectivity of the dBG has been extensively studied by many researchers. The following are some papers addressing this issue.

- M.Imase *et al.* [IMAS85] derived two inequalities of the edge and node connectivity c_e and c_n of the dBG in terms of number of nodes and maximum and minimum degree. From these results, it was shown that diameter minimization results in maximizing the connectivity.
- A.Esfahanian *et al.* [ESFA85] improved further on [PRAD82]. The paper showed that the node-connectivity of dBG is 2Δ-2 instead of r in [PRAD82]. This would imply that the system can tolerate up to 2Δ-3 node failures instead of Δ-1. The routing methods for situations with a certain number of node failures were also presented and they are shown to be computationally efficient.
- M.A.Sridhar published in past few years a series of papers on the connectivity and the fault tolerance of the dBG[SRID88, SRID91, SRID92]. He used the known results about periods in strings to simplify the derivation and tighten the bounds derived by previous researchers. It was shown that in the presence of up to 2Δ-3 faulty nodes, the increase in the diameter of this graph is at most $\log_k \log_{\varnothing} m+6+\log_k 5$, where $\varnothing$ is the golden ratio. Sridhar and Raghavendra also introduced a new class of networks based on the dBG which can tolerate up to Δ-2 node faults.

The dBG has also been used in various contexts, including sorting networks[SAMA89], permutation networks[SRID87] and parallel architecture[BERM89, ROWL93]. More recently, however, as the idea of multichannel multihop system has been developed, researchers are attracted by the unique properties of the dBG which has a considerably small mean path length and the ability of accommodating a large number of nodes. The study of the multihop systems based on the dBG has been addressed in [SIVA91, SIVA92] and summarized in [MUKH92b, GREE93].

In [SIVA91], unidirectional dBGs have been proposed as logical topologies for multihop lightwave networks. The throughput and delay performance of networks are compared with those of ShuffleNets. A simplified delay analysis based on M/M/1 queueing models for links and independence assumptions indicates that, for uniform loading, the average packet delay in a dBG can be slightly lower than that in an equivalent

ShufflenNet for low to moderate load. It was also shown that, as mentioned above, a dBG can support a larger number of nodes than can ShuffleNets.

Through the delay and throughput analysis of dBG networks, K.N.Sivasrajan[SIVA92] shows that the multihop architecture is an attractive option today, and will remain so until we find low-cost fast-tunable optical components, and efficient but practical solutions to the multiple access problem.

A recent paper[SIVA94], which has just been received while this thesis is underway, has a more complete study on the lightwave networks based on the unidirectional dBGs. Two different routing schemes have been proposed for such networks, i.e., Shortest-path and Longest-path Routing. Network throughput and packet delay performance effected by these two methods have been studied. Comparison with the ShuffleNets has also been made. The paper also proposed dBG as good physical topologies for wavelength routing lightwave networks. The optical implementation of such networks is proposed. The difference of the paper and this thesis is that the latter focuses on the bidirectional dBG and its routing algorithms, the hierarchical structures of dBGs and the detail optical node structures which are not discussed in the mentioned paper.

1.7 Motivation And Approaches Of The Thesis

There have been a lot of studies on designing structures for optical multihop systems. Among all proposed structures for multihop lightwave systems, the de Bruijn Graph(dBG) network has proved to be a competitive candidate as discussed earlier. It is shown that it has comparable or even better performance than ShuffleNets[SIVA91]. Furthermore, dBG networks also admit very simple routing algorithms which can lead to shortest path routing in unidirectional case. Therefore, we are interested in considering dBG as a candidate topology for the next generation of networking and evaluating its performance.

Although some works have been done on the study of the topology of dBG, it has not been studied from the optical network point of view as a topology for multihop systems until [SIVA91]. As we understand, most of the previous works concentrated on the connectivity issue and fault-tolerance routing. Yet there are two important issues which the system architect must address when designing a "good" structure[MUKH92b]. First, the virtual structure chosen must be close to optimal in some sense; for example, the structure's average hop distance(mean path length) between nodes must be small, or the average packet delay must be minimal, or the maximum flow on any link in the virtual structure must be minimal. Second, the nodal processing complexity also must be small because the high-speed environment allows very little processing time; consequently, simple routing mechanisms must be employed.

Therefore, to find simple routing algorithms that would effectively improve the network performance motivates the study of this thesis. There is little literature about the multihop systems based on the Bidirectional de Bruijn Graph(BdBG), especially about the routing algorithms in the BdBG. In fact, there is no known simple way to generate a routing of shortest paths in the BdBG[BERM89]. To our best knowledge, all researchers used so-called Shortest Consistent Path routing algorithm while studying the BdBG networks. The purpose of this thesis is to propose various routing algorithms that would lead to shorter mean path length of the BdBG networks and also to study their effect on the performance of the networks.

Finally, in order for dBG network to be a successful candidate for the next generation of networking, it should yield an optical implementation. So far there is not much coverage in this aspect. Therefore, we are interested in studying the feasibility of its implementation by optical devices.

In pursuing our interests above, we shall use the dBG as a multihop optical network, and evaluate the performance in its logical topological form, especially the bidirectional dBG which has been mentioned little in the literature. There are different measures to analyze and compare the effectiveness of multihop systems. In this thesis, we use the mean

path length because it is a function of topology and routing algorithms. Different routing algorithms have been coded by computer programs in order to observe their performances.

In designing a Metropolitan Area Network(MAN), one must address the issue of accommodating large number of nodes, say more than one thousand, and covering a large area, say tens of kilometers. This may become an undesirable factor for regular one-level networks which may require many interconnections from end to end. That leads us to consider hierarchical structures which can reduce a large amount of link cost and also are more adaptable to future network growth. Moreover, hierarchical structures can often exploit the locality in communication and reduce the average propagation time of the messages; therefore, the network performance can be improved. These effects of the hierarchical structures by the locality and the propagation time will be studied later in this thesis.

Mean value analysis will be carried out from derived equations. Parameters in the equations are obtained by computer programs. In cases, such as mean path length of the network, when closed form expressions can not be found, computer emulation is used to obtain the values. The traffic intensity from which the network delay and throughput can be derived is also obtained by means of computer programs according to different routing schemes.

The experience gained from the analysis, such as complexity analysis of the routing algorithms, bridge choosing in the hierarchical networks, is used to provide design guidelines. Finally, implementation using optical devices is introduced since it is an important part in design.

1.8 Thesis Organization And Contributions

The thesis is organized as follows.

- Chapter 2 studies the construction of the network based on the dBG and also proposes three new routing algorithms used in the Bidirectional dBG networks. Network per-

formance in terms of mean path length, delay and throughput is studied. Comparisons between unidirectional and bidirectional dBGs, Shortest Consistent Path routing(proposed by previous researcher) and three new routing schemes are provided. The complexity of routing methods is also addressed.

- Chapter 3 proposes hierarchical structures of unidirectional and bidirectional dBGs. Method of choosing bridge nodes is discussed. Network performance based on three different situations is studied; i.e., uniform traffic case, consideration of locality and propagation time.
- Chapter 4 shows the general optical node structures for regular nodes and bridge nodes. Network design issue is discussed. A general design procedure is also proposed.
- Chapter 5 concludes the thesis, and suggests some potential areas for future work.
- The Appendix is composed of programs from which research results are derived.

The main contributions of the thesis are listed in the following.

- Proposing three new routing algorithms in the Bidirectional de Bruijn Graph networks and studying their performance. Necessary comparisons with previous results are also given.
- Estimating the complexities of three new routing algorithms and providing the basis for network designs.
- Proposing the hierarchical structures of unidirectional and bidirectional de Bruijn Graph networks as metropolitan area lightwave networks.
- Studying the method of choosing appropriate bridge nodes in the hierarchical structures.
- Evaluating the hierarchical network performance under three different circumstances: uniform traffic distribution, presence of locality factor and propagation time.
- Using optical devices to implement regular and bridge nodes.
- Discussing the design issue of networks based on de Bruijn graph.

1.9 Publications

Parts of the research work in this thesis have been reported in the following papers.

- Zhi Feng and Oliver W.W. Yang, "Metropolitan Area Networks Based on the Hierarchical Unidirectional and Bidirectional de Bruijn Graph," to appear in *Proc. of the 28th Annual Conference on Information Science and Systems,* Princeton University, Princeton, NJ., Mar. 1994.
- Zhi Feng and Oliver W.W. Yang, "Routing Algorithms In The Bidirectional De Bruijn Graph Metropolitan Area Networks," to appear in *Proc. of 1994 IEEE Military Communications Conference.* New Jersey, Oct.1994.
- Oliver W.W. Yang and Zhi Feng, "DBG MANs and their Routing Performance," submitted for publication.

Chapter 2

DBG Networks and Their Routing Algorithms

2.1 Introduction

In this chapter, the structures of the Unidirectional de Bruijn Graph(UdBG) and Bidirectional de Bruijn Graph(BdBG) networks are discussed. Three new routing algorithms that can be used to improve the performance in BdBG networks are proposed. Studies on the comparisons among these routing methods and the previous proposed method are also performed. The complexity issue of routing algorithms is addressed. Network performance is shown in terms of the mean path lengths, average packet delay and network throughput parameters.

2.2 The UdBG and BdBG Networks

2.2.1 Constructing the UdBG Networks

A (Δ,D) Unidirectional de Bruijn Graph(UdBG), also called directed de Bruijn Graph, is a graph with the address of each vertex represented by a string of D Δ-ary digits, where Δ and D are positive integer larger or equal to 2. The graph contains Δ^D vertices. The graph can be constructed by providing a directed link from a vertex with address (A_D, A_{D-1},..., A_1) to another vertex with address (B_D, B_{D-1},..., B_1) that satisfies the following condition only:

$$B_i = A_{i-1}; \quad \text{where } B_i, A_i \in \{0,1,2,\ldots,D-1\} \text{ and } 2 \le i \le D.$$

i.e. $(A_{D-1}, A_{D-2}, \ldots, A_1)$ are left shifted one digit to become $B_D, B_{D-1},\ldots, B_2$. Note that B_1 can be replaced by any digit from 0 to Δ-1.

Vertex $(B_D, B_{D-1},\ldots, B_1)$ is called a *left neighbor* of vertex $(A_D, A_{D-1},\ldots, A_1)$. Note that the link from $(A_D, A_{D-1},\ldots, A_1)$ to $(B_D, B_{D-1},\ldots, B_1)$ is unidirectional, so that there is no link connecting from $(B_D, B_{D-1},\ldots, B_1)$ to $(A_D, A_{D-1},\ldots, A_1)$. Consequently, there is no right neighbors at each vertex. An example of a (2,3) UdBG has been shown in Figure 1.5 in which node 010 and 011, for instance, are the left neighbors of node 001.

The maximum degree of the UdBG is Δ; i.e., every node(vertex) has Δ incoming links and Δ outgoing links except the nodes with address (A, A, ..., A) where $A \in \{0,1,2,\ldots,D-1\}$. We call these special nodes as *extremity nodes*; for instance, nodes 000 and 111 in Figure 1.5. The self-loops at the extremity nodes are often neglected in real networks, so they are different from other ordinary nodes in the sense that their degrees are Δ-1.

2.2.2 Constructing the BdBG Networks

An interesting observation is that we can form a Bidirectional de Bruijn Graph(BdBG) by changing all the unidirectional links to bidirectional links in the UdBG while keeping the original addressing scheme of the network. Mathematically, a vertex with address $(A_D, A_{D-1},\ldots, A_1)$ will connect to vertex $(B_D, B_{D-1},\ldots, B_1)$ if one of the following two conditions are satisfied:

1. $B_i = A_{i-1}$, where $B_i, A_i \in \{0,1,2,\ldots,\Delta-1\}$ and $2 \le i \le D$;
2. $B_i = A_{i+1}$, where $B_i, A_i \in \{0,1,2,\ldots,\Delta-1\}$ and $1 \le i \le D-1$.

Due to this property, a BdBG is alternately called the shift-and-replace graph in the sense that the addresses of any two connected nodes are one digit different from another after shifting the address either to left or to right by one digit. We shall use this observation to construct our new network.

An example of a (2,4) BdBG network is shown in Figure 2.1. The solid lines represent the original unidirectional links, while the dotted lines are the links in the opposite direction. It can be observed that the dotted lines form another UdBG network whose node addresses have bits ordered in the reverse direction. For example, nodes 0001 and 0011, which are connected by a solid unidirectional link, take on the addresses 1000 and 1100 respectively when connected by a dotted unidirectional link. In another word, a node address takes on two representations: one for the solid-line UdBG and the other for the dotted-line UdBG network.

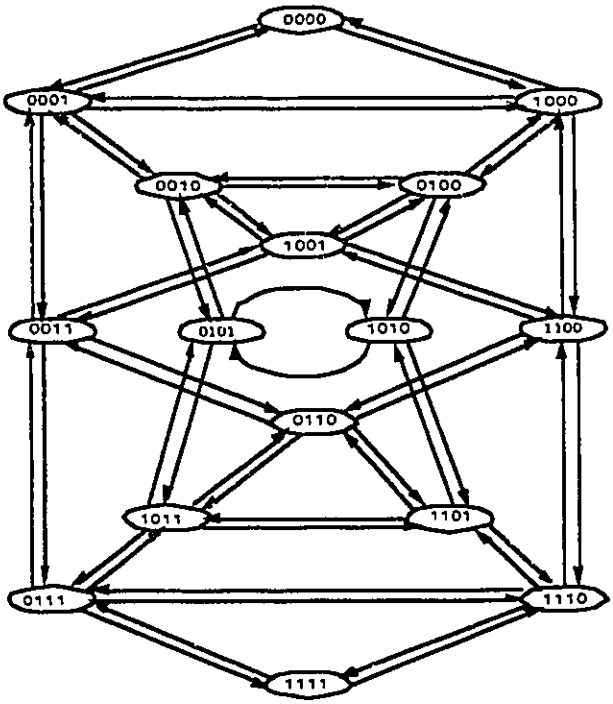


Fig.2.1: A (2,4) BdBG network. (redundant links are omitted)

We call vertices satisfying condition 1 as the *left neighbors* of (A_D, A_{D-1},..., A_1), and those satisfying condition 2 as the *right neighbors* of (A_D, A_{D-1},..., A_1). In a BdBG network, every node is connected to its left neighbors as well as its right neighbors. As a result, the maximum degree of the network is doubled(i.e., 2Δ) while the minimum degree is 2Δ-2 [SRID92]. Like the UdBG, the BdBG has Δ^D nodes, and its diameter D, defined

as the maximum hops between any node pairs, is the same as the UdBG because at least it can be seen that the distance between two extremity nodes does not change with respect to its unidirectional counterpart.

2.3 Routing Algorithms in dBG Networks

2.3.1 Routing Algorithm in the UdBG Networks

Routing in the UdBG is simple. Notice that a link between two adjacent nodes can be represented by (D+1) Δ-ary digits, the first D digits represent the source node and the last D digits represented the destination node. For example the link between node 001 and 011 in Figure 1.5 can be represented as 0011. In a similar fashion, any path of length k can be expressed by D+k digits.

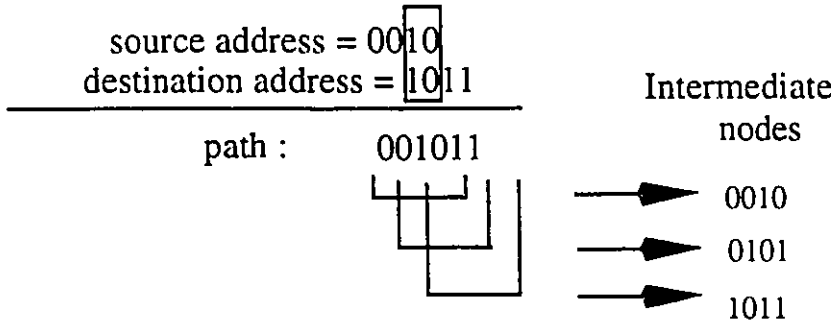
To determine the shortest path from node $A=(A_D, A_{D-1},\ldots, A_1)$ to node $B=(B_D, B_{D-1},\ldots, B_1)$, one only needs to consider the last several digits of A and the first several digits of B to obtain a perfect match over the largest possible number of digits. If this match is of size of k digits, i.e.,

$$(A_k A_{k-1}\ldots A_1) = (B_D B_{D-1}\ldots B_{D-k}),$$

then the shortest path would be $D-k$ hops and its path would be

$$(A_D A_{D-1}\ldots A_1 B_{D-k-1} B_{D-k-2}\ldots B_1).$$

The intermediate nodes can also be determined by the matched pattern. The more digits it matches, the shorter the path is between them. For example, in a (2,4) UdBG, if node 0010 wants to transmit a packet to node 1011, because "10" is the postfix of the source and the prefix of the destination, node 0010 knows that it needs two hops to reach node 1011 and also the intermediate nodes the packet will go through as shown in the following illustration.



Therefore, each node only has to know the address of the destination node stored in the header of the incoming packet to determine the outgoing path the packet should take. This algorithm gives a fixed path for every source-destination node pair, and produces a shortest-path according to the property of the UdBG.

2.3.2 Routing Algorithms in the BdBG Networks

In a BdBG, since a packet transmitted from a node can be sent to either its left neighbors or its right neighbors, we define the **Forward Path(FP)** to be the path taken by a packet when a left neighbor is always chosen as the next node to send from its current node. For example, the solid lines in Figure 2.1 represent the FPs. On the other hand, packets travel along the **Backward Path(BP)** when they always choose a right neighbor till they arrive at the destination node. Note that if only the FPs are taken in the BdBG, it becomes exactly the same as the UdBG because only the solid-line network in Figure 2.1 is used. Our routing algorithms are formulated based on the above observations.

Considering taking only the FP or BP, a node can calculate the number of hops that a packet needs to reach the destination. This is done as in the UdBG shown above by matching the postfix portion of the source address with the prefix portion of the destination address if FP is used. If BP is used, one needs to reverse the node address before doing the matching. For example, node 0101 should be read as 1010 in Figure 2.1.

Unlike the UdBG network, a shortest-path algorithm for a general BdBG network has not been found. We propose in this section three possible routing algorithms that have

improved on their capability in seeking out the shortest path. They are respectively called the Refreshing(RFR), Neighbor Searching(NSC) and Pattern Matching(PMC) algorithms, and will also be compared with the existing routing method [EAFA85] referred to the Shortest Consistent Path(SCP) algorithm which is also shown here for comparison purpose.

Before we introduce these algorithms, we shall define fwd(Sc, De) and bwd(Sc, De) to be respectively the forward path and the backward path from Sc to De, where Sc and De are the source and the destination address. Define also **match_fwd(Sc, De)** to be an operation which returns the number of hops required to reach the destination along a FP. Similarly, **match_bwd(Sc, De)** returns the number of hops along a BP. Then the four routing algorithms can be described as follows.

1) The Shortest Consistent Path(SCP) Routing Algorithm

The SCP algorithm is the simplest but it does not guarantee a shortest path. Basically, a node can determine the length of the shortest path in either one of the two UdBG networks (i.e., the FP UdBG or the BP UdBG) that make up the BdBG network. Then it will choose the UdBG that has a shorter path length. Once the path is chosen at the source node, the packet will not change its direction at the intermediate nodes. It simply follows the same path as in the unidirectional case. The algorithm can be described as following with Pr representing the present node's address.

```
begin
   if (De == Pr ) receive the packet;
   if (Sc ≠ Pr ) following the path fixed by the source;
   else begin
        FP length  i = match_fwd(Sc, De);
        BP length  j = match_bwd(Sc, De);
           if ( i < j )  path = fwd(Sc, De);
           else if ( i == j )  path = randomly choose fwd(Sc, De) or bwd(Sc, De);
           else        path = bwd(Sc, De);
        end
end
```

Note that this routing algorithm can be advantageously used in source routing by embedding in the packet the direction (FP or BP) a packet has been assigned to.

2) The Refreshing(RFR) Routing Algorithm

Although the SCP algorithm is quite simple to use, it does not take much advantage of the bidirectional links in the BdBG. This is because a packet might miss a shorter path if it is not allowed to change the direction (from FP to BP or vice versa) at the intermediate nodes. For example in Figure 2.1, if node 0001 has a packet to send to node 1001, match_fwd(0001,1001) produces a path length of 3 hops in the FP-network, while match_bwd(0001,1001) produces 4 hops in the BP-network. The packet, therefore, will choose a FP and travel along the node route (0001-0010-0100-1001) which has 3 hops. By inspection, however, one can see that the packet should be sent along the node route (0001-0010-1001) which has 2 hops; this requires the packet to switch from the FP to BP at node 0010. In view of this shortcoming, the RFR routing algorithm is formulated as in the following.

```
begin
  if (De == Pr ) reccive the packet;
  else begin
       FP length from the present node:  i = match_fwd(Pr, De);
       BP length from the present node:  j = match_bwd(Pr, De);
       if ( i < j )  using FP send packet to the left neighbor;
       else if ( i == j )  randomly using FP or BP ;
       else       using BP send packet to the right neighbor;
  end
end
```

The difference between SCP and RFR is that in RFR, decision is a function of the present and the destination addresses instead of the source and the destination addresses in SCP. Upon arriving at a node each time, the packet is "refreshed" in the sense that it would be treated as if the packet is sent by this present node. It checks the distance to the destination and then decides whether it will send the packet along the FP or the BP.

Consequently, the packet may change direction within the transmission, and it certainly will shorten the path lengths on the average. This will be demonstrated later.

3) The Neighbor Searching(NSC) Routing Algorithm

Although the RFR has improved the mean path length performance by allowing a packet to change its direction at intermediate nodes, it does not always produce the shortest path. This is because in some cases, a neighboring node that was judged to be on a longer path by the RFR algorithm might actually produce a shorter overall path length. In other words, the RFR algorithm is not aware of the distance from the next node to the destination. For example, if node 1001 in Figure 2.1 wants to transmit a packet to node 1000, it would obtain FP(1001,1000)=3 and BP(1001,1000)=4. Node 1001, therefore, would send the packet along FP to node 0010 and it takes in total 3 hops to reach node 1000. However, if node 1001 uses a BP to send to node 0100, and then a FP to node 1000, a packet will take only 2 hops.

The above consideration leads to the Neighbor Searching(NSC) Routing Algorithm which is formulated as follows.

```
begin
  if (De == Pr ) receive the packet:
  else begin
  counter=1
  while (counter ≤ number of neighbors) do
          begin
       i=match_fwd(address of neighbor_counter, De);
       j=match_bwd(address of neighbor_counter, De);
       required path length at neighbor_counter
       path[counter]= minimum ( i, j );
              counter=counter+1;
          end
   next node = one of the neighbors with minimum path[counter];
   send packet to the next node ;
  end
end
```

Since every node knows the addresses of its neighbors, it can actually determine the number of hops required to go from its neighbors to the destination node in the FP or BP. Then it sends the packet to the neighbor which has the minimum required hops to the destination. Thus this algorithm allows a node to make a smarter decision on the next node to send by exploiting the property of the BdBG. Although this still does not produce the shortest path , it will be shown later that the mean path length is nevertheless shortened.

4) The Pattern Matching(PMC) Routing Algorithm

The PMC routing algorithm takes advantage of the inherent relation between the address bit patterns of the present location and the destination. In general, any pair of addresses might have certain bit pattern in common. Intuitively, the more common digit pattern matches between the addresses of source and destination, the smaller the distance is between them. Thus the idea behind Pattern Matching is to find out the criteria that can orient the packet towards the destination.

One natural criterion is the Width of the Common Pattern(WCP) defined to be the number of digits of the largest pattern that matches two addresses. The larger the WCP is, the closer are the two nodes. For example, node 0011 is closer to node 0110 than node 0001 because the WCP of (0011, 0110) is 3 while the WCP of (0001, 0110) is 2. In the special case of the source equal to the destination, the addresses match completely and their WCP attains the largest value.

Since this criterion alone is not sufficient to always orient the packet towards its destination, we introduced another criterion called the Distance of the Common Patterns(DCP), defined to be the difference in the positions of the two common patterns. In the example given above, the DCP of (0011, 0110) is 1 as their common pattern "011" is at an offset of one position from the other. Likewise the DCP of (0001, 0110) is 2. Using the addressing property in de Bruijn graph, one can verify that for a given common pattern, the larger the DCP, the closer are the source and destination nodes to each other. For example, CPs of both (0011, 0110) and (0111, 0110) are 3, and their DCPs are 1 and 0 respectively. Therefore, node 0011 is closer to node 0110 than node 0111.

Having defined these two criteria, we now introduce in the following the NSC routing algorithm between the present node(Pr) and the destination node(De).

```
begin
  if (De == Pr ) receive the packet;
  else begin
       WCP_current = WCP of (Pr , De);
       DCP_current = DCP of (Pr , De);
       while (there is an unchecked neighbor) do begin
           WCP_temp = WCP of (this neighbor , De);
           DCP_temp = DCP of (this neighbor , De);
           if (WCP_temp > WCP_current ) begin
               next node = this neighbor ;
               WCP_current = WCP_temp ;
               DCP_current = DCP_temp ;
           end
           else if (WCP_temp == WCP_current ) begin
               if (DCP_temp > DCP_current ) begin
                   next node  = this neighbor ;
                   DCP_current = DCP_temp ;
               end
           end
       end
               send packet to the next node  ;
    end
end
```

In essence, the algorithm uses the criteria to determine which neighbor is closer to the destination in the sense that its address matches more with the destination address. Notice that this pattern matching technique can make use of any other criteria that can correctly orient the packet. For example, when two neighbor nodes have the same WCP and DCP, a criterion can be added to so that the packet will pick either one randomly.

2.4 Performance Analysis of the UdBG and BdBG Networks

As our approaches stated in Chapter 1, we shall use the mean path length, average packet delay and network throughput parameters to illustrate the network performance. All these parameters vary depending on the routing algorithm applied. A routing protocol which causes traffic to be transferred along longer paths will generate more internal traffics and the derived mean path length will be larger accordingly. It is also very important to determine the mean path length of a multihop lightwave system due to the fact that at each hop there is opto-electronic signal processing which tends to be the bottlenecks in very high speed networks. Consequently, one would prefer a network that has a small mean path length.

2.4.1 Mean Path Length(Average Number of Hops)

Define mean path length $\overline{H}$ as the average number of hops among all source destination pairs in the network. Mathematically, it is given by $\overline{H} = \frac{1}{N(N-1)} \sum_{i=0}^{D} i\, n(i)$, where $n(i)$ is the number of node-pairs in the network with a path length of i hops, N is the number of nodes in the network and D is the network diameter and also the number of digits of the node address.

The mean path length can also be found from the ratio of the total internal traffic carried in all links of the networks to the total external traffic generated at their sources[KLEI64], i.e.,

$$\text{mean path length} = \frac{\text{total internal traffic}}{\text{total external traffic}} \qquad (2.1)$$

Computer programs have been coded in C programming language so that the internal traffic intensity(also called edge load which will be defined later) at each link can be obtained. The four routing algorithms applied to BdBG networks are emulated in different programs listed in the Appendix A. In each program, external traffic is injected between a source and a destination node and the resulted internal traffic of each link along the path (decided by a particular routing algorithm) is accumulated. After repeating this process for every source-destination pair, all internal traffic can be summed up. Therefore, the external traffic is simply N(N-1), where N is the number of nodes in the network. Consequently, we are able to obtain the mean path length parameters by using the above ratio.

Table 2.1 shows the derived mean path lengths of the BdBG using the 3 proposed routing schemes. The performance of SCP is also tabulated for comparison purpose. Note that for small networks, no differences exist in terms of the mean path length among all routing algorithms since SCP has already achieved the shortest path. For other networks, the performance has been improved by RFR and further by NSC; this is because NSC checks one step ahead. For larger networks, PMC produce shorter paths in most low degree cases; however, in some higher degree cases, the mean path length obtained by PMC may be larger.

In order to relate the BdBG networks with other architectures and therefore obtain better knowledge of their performances, we have chosen the Bilayered ShuffleNets proposed in [AYAD93] which have been shown to achieve much higher efficiency compared to the conventional ShuffleNets. To make the comparison more meaningful, we have only chosen several examples in which the BdBG networks and the Bilayered ShuffleNets have the same degree and number of nodes. The NSC routing algorithm which produced shorter mean path lengths is used in the BdBG networks. Results are listed in Table 2.2. It shows that the BdBG networks are commensurate with the Bilayered ShuffleNets. Notice that the mean path lengths of the Bilayered ShuffleNets shown in Table 2.2 are obtained by assuming shortest routing algorithm, while those of BdBG networks can be improved further as the NSC routing scheme does not necessarily produce a shortest path.

Table 2.1: Number of nodes and mean path length as a function of Δ and D in BdBG networks.

Δ, D	No. of nodes	Mean Path Lengths			
		SCP	RFR	NSC	PMC
2, 2	4	1.167	1.167	1.167	1.167
2, 3	8	1.643	1.643	1.643	1.643
2, 4	16	2.258	2.188	2.146	2.142
2, 5	32	2.984	2.796	2.794	2.766
2, 6	64	3.801	3.653	3.551	3.495
3, 2	9	1.417	1.417	1.417	1.417
3, 3	27	2.128	2.105	2.007	2.077
3, 4	81	2.978	2.911	2.865	2.849
3, 5	243	3.907	3.800	3.755	3.736
3, 6	729	4.875	4.775	4.704	4.722
4, 2	16	1.55	1.55	1.55	1.55
4, 3	64	2.369	2.343	2.321	2.321
4, 4	256	3.298	3.251	3.214	3.218
4, 5	1024	4.273	4.207	4.172	4.209
5, 2	25	1.633	1.633	1.633	1.633
5, 3	125	2.510	2.491	2.471	2.471
5, 4	625	3.471	3.438	3.41	3.437
6, 2	36	1.690	1.690	1.690	1.690
6, 3	216	2.601	2.585	2.570	2.570

Table 2.2: Comparison between BdBG networks and Bilayered ShuffleNets.

Degree of the network	No. of nodes in the network	Mean path lengths	
		Bilayered ShuffleNets	BdBG networks using NSC
4	8	2.0	1.643
4	64	3.651	3.551
8	1024	4.174	4.172

2.4.2 Edge Loading, Delay and Throughput Performance

In performance evaluation, one is usually interested in the maximum throughput a network can support. Since any communication path is merely a concatenation of links in a network, the maximum data rate a link can support would have an impact on network throughput. To investigate this, define Edge loading to be the amount of traffic present at one link in response to external traffic offered to the network. Note that even if the offered traffic to the BdBG network is fully symmetric, the edge loading can be unequal at each link as in the dBG networks[MUKH92b]. As a result, the system throughput is limited to the maximum edge load of the network. Define the normalized network throughput to be equal to the reciprocal of the maximum edge load. By using the same programs which enumerate the internal traffic in the network, the maximum edge load and hence the normalized throughput can also be obtained.

Table 2.3: Maximum edge load and Throughput performance of UdBG and BdBG networks

Δ, D	No. of nodes	Maximum edge load		Throughput		Improvement factor
		UdBG	BdBG	UdBG	BdBG	
2, 4	16	29	13	0.0344828	0.0769231	2.231
2, 5	32	81	36	0.0123457	0.0277778	2.250
2, 6	64	208	85	0.0048077	0.0117647	2.447
3, 4	81	138	57	0.0072464	0.0175439	2.421
3, 5	243	535	217	0.0018692	0.0046083	2.466
3, 6	729	1945	767	0.0005141	0.0013038	2.536
4, 4	256	313	150	0.0031949	0.0066667	2.087
4, 5	1024	1589	719	0.0006293	0.0013908	2.210
5, 4	625	586	291	0.0017065	0.0034364	2.014

Table 2.3 gives the maximum edge loading and throughput performance of various (Δ,D) dBG networks. The throughput improvement factor, defined as the ratio of the

throughput of BdBG networks and the one of UdBG networks, are also presented. The results in Table 2.3 show that the BdBG networks improve the total throughput by more than twice and therefore justify the efficiency of the BdBG networks.

It is also observed that with the same degree Δ, as the diameter of the network increases, the improvement factor is increasing as well. However, with the same network diameter D, increasing the network degree Δ does not necessary produce a better improvement. For example, the (2,6) BdBG network has an improvement factor of 2.447 while the (2.5) BdBG network has 2.250 only, but the (4,5) BdBG network only has 2.210 improvement factor.

The end-to-end delay from the source to the destination node can be considered as the sum of the queuing delays and the propagation delays along the path. In evaluating the average end-to-end delay, we take the following assumptions to simplify the analysis:

- Offered load to each node is symmetric and packets have equal probability of being destined to any other nodes in the network. We shall further refer this as the uniform traffic case throughout this thesis.
- Packet arrivals to each node are Poisson with a mean rate of λ packet/s.
- The packet lengths are exponentially distributed with a mean length of $1/\mu$ bits.
- Each node has an infinite buffer.
- The propagation delay is equal at each link.

Under these assumptions, we can model each queue as independent M/M/1 queues to analyze the system[KLEI75]. Let C be the capacity of a link in bits/s, L_i be the edge load at link i, N is the number of nodes in the network, J is the number of links in the network and τ is the propagation delay at each link, then the mean queuing delay $\overline{D}$ for a packet is given [SIVA91] as

$$\overline{D} = \frac{1}{N\ (N-1)} \sum_{i=1}^{J} \frac{L_i}{\mu C - L_i \lambda} + \overline{H} \cdot \tau. \qquad (2.2)$$

If the propagation delay is assumed to be 0, then Equation (2.2) can be reduced and then rearranged as

$$\overline{D}\,\mu C = \frac{1}{N\ (N-1)} \sum_{i=1}^{J} \frac{L_i}{1 - L_i \cdot \lambda/\mu C}. \qquad (2.3)$$

Equation (2.3) shows the relationship of the normalized delay $\overline{D}\,\mu C$ and the normalized offered load $\lambda/\mu C$ without considering the propagation delay. Therefore, we can have the curve of the normalized delay by just obtaining the traffic load at each link in the network. To obtain the real packet delay in second in a particular network, one only needs to multiply the normalized delay by a factor of μC.

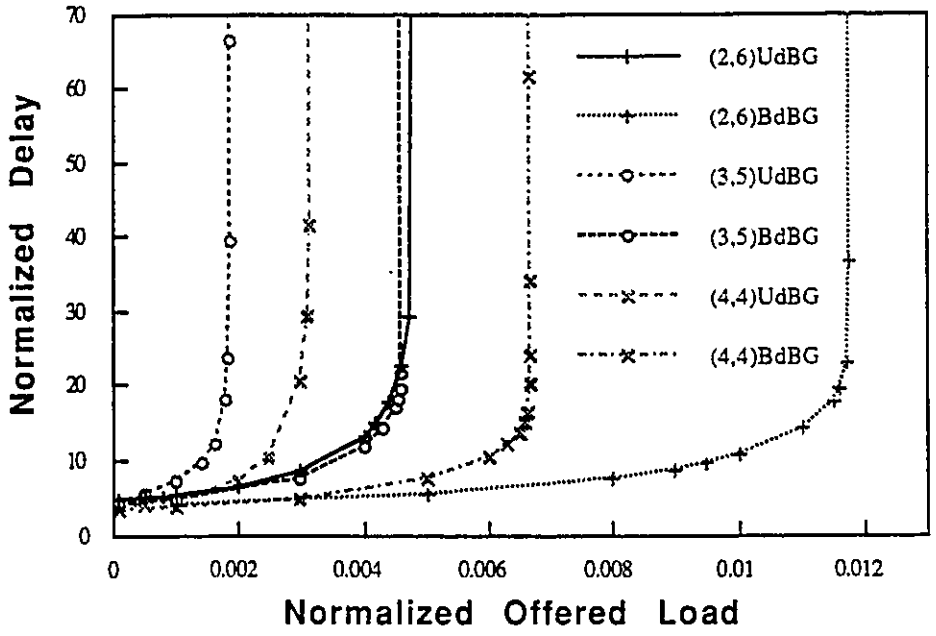


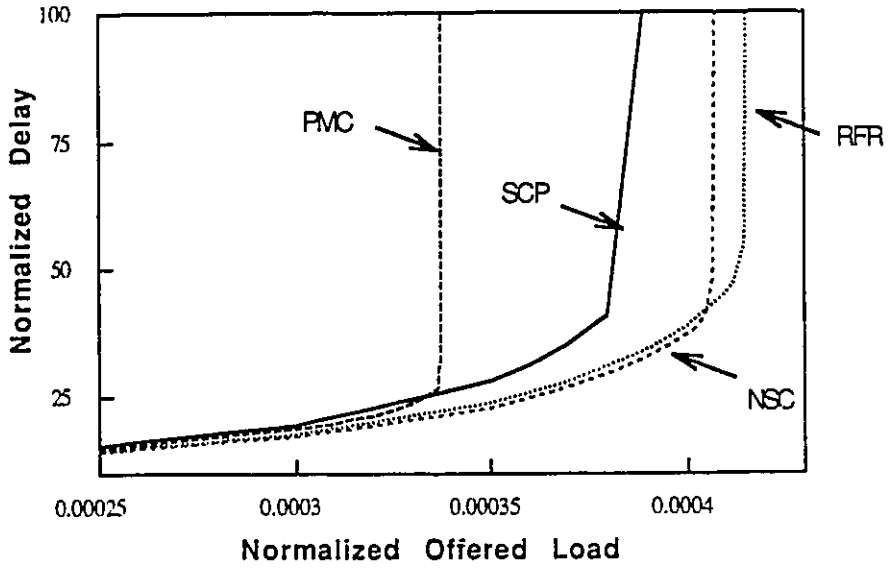
Fig. 2.2: Comparison between UdBG and BdBG networks.

Figure 2.2 shows such an example of normalized delay $\overline{D}\,\mu C$ versus a normalized offered load $\lambda/\mu C$ for the comparison between UdBG and BdBG networks. Three groups of networks with (Δ,D)=(2,6), (3,5) and (4,4) respectively are presented. RFR routing is assumed in BdBG networks. From the graph we can see that the maximum normalized throughput of the network, which is the value of normalized offered load in the graph when

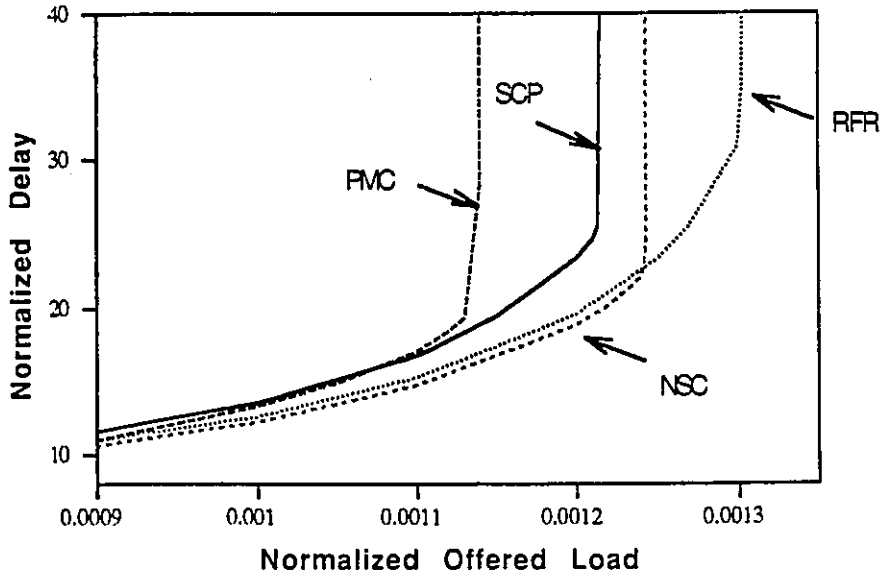
delay goes to infinite, equals to the reciprocal of the maximum edge loading. At this point, the denominator inside the summation in Equation (2.3) approaches to 0. For example, the maximum normalized throughput of the (2,6) BdBG network shown in the graph is 0.012 while the (2,6) UdBG is about 0.005. It is observed that the BdBG network not only has doubled the connectivity between all nodes, but has also more than doubled the throughput. Similar observations can be made in other (Δ,D) BdBG networks. We leave them to the reader verify that. This justifies the efficiency of the BdBG networks. In addition, Figure 2.2 shows the BdBG networks have lower delay in the entire region of the offered load compared to their UdBG counterparts because of the shorter mean path length they have.

The delay comparison among the 3 proposed routing schemes along with the SCP as reference is provided in Figure 2.3. An example of (2,10) BdBG network with 1024 nodes which is about the size of a metropolitan area network is shown in Figure 2.3(a). Figure 2.3(b) shows a (3,6) network which has 729 nodes. In both graphs, it is observed that in the low offered-load region, the normalized delay achieved by each routing algorithm is nearly the same as the mean path length because it is the average number of hops that packets go through. As the load increases, their delays blow up at different points. It is interesting to see that the RFR and NSC algorithms are more promising because they can achieve relatively smaller delay but higher throughput in the high offered load region. The maximum throughput of the RFR is better than the NSC, but its delay is actually slightly larger than the NSC for most of the offered-load ranges. On the other hand, the PMC has a lower delay than that of the SCP, but it offers the lowest throughput capability.

To be more realistic, the performance of a network with typical parameter values is presented in Figure 2.4. The network has the same configuration as the one in Figure 2.3(a). Its propagation delay at each link τ =250μs, 1/μ=10,000 bits, and C=1Gb/s. Figure 2.4 shows the delay and the throughput performance of the network using different routing algorithms.



(a): A (2,10) BdBG network.



(b): A (3,6) BdBG network.

Fig. 2.3: Normalized delay and throughput performance of four routing algorithms assuming $\tau = 0$.

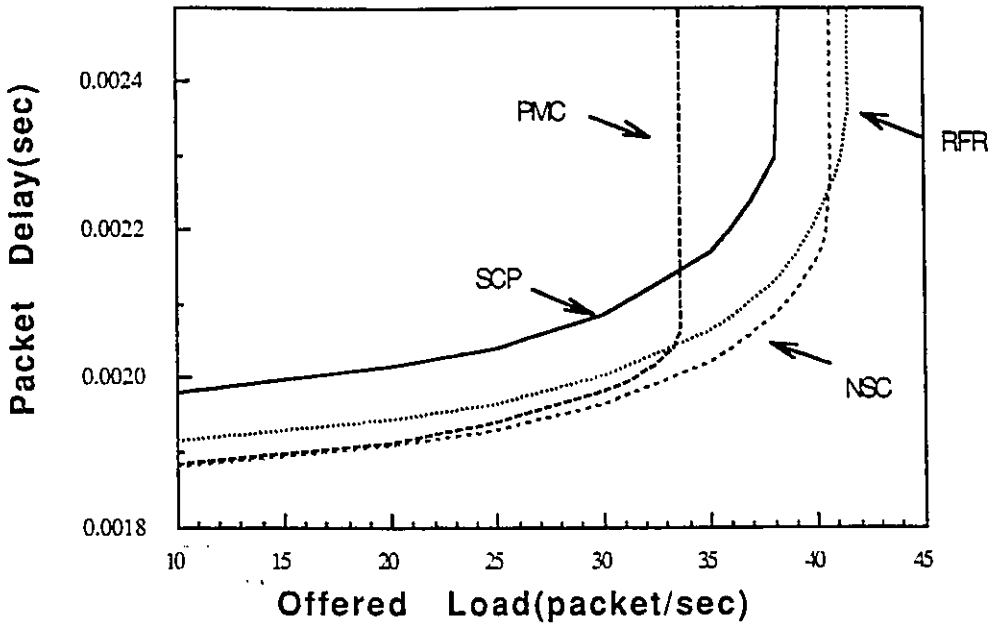


Fig. 2.4. Delay and throughput performance of four routing algorithms in a (2,10) BdBG network when τ = 250μs.

It is obvious that because of the effect of the propagation time on each link, the differences among the curves of the packet delay are enlarged. This can also be seen from Equation (2.2). The propagation time is weighted by the factor of mean path length; therefore, as the propagation time increases, the one with larger mean path length will have more increase of the packet delay. Consequently, the difference among the delay curves in Figure 2.4 is enlarged compared to Figure 2.3a in which the normalized delay can be converted to the actual delay value by multiplying it by μC.

In the low offered load region, the SCP has the highest delay because of its highest mean path length among all four routing methods. The PMC still has the lowest throughput performance; however, the crossover point of the PMC with the SCP is moved to the right. It means that the PMC performs better when the propagation time becomes larger. From the figure we can see that the RFR and the NSC methods tend to be more

promising because their considerable low delay in the low offered load region and the high throughput the network can support.

2.5 Complexities of Routing Algorithms

The complexity is another interesting measure which one can use as comparison among the algorithms; consequently, one would realize the tradeoff in their implementations.

The complexity discussed in this section is determined by the number of operation steps a node has to carry out when executing a particular routing algorithm.

The SCP method decides the whole path at the source node and the packet direction can be embedded in its header so that the intermediate nodes can have the knowledge of the next node. Therefore, the complexity only comes from the source when doing the address matching. Basically it is to find out the largest matched portion between the postfix of the source address and the prefix of the destination address. Notice that this decision mechanism is repeated D times if the address has D digits. Consequently, the complexity is O(D) where D is the length of the addresses.

The RFR method does exactly the same mechanism as the SCP except that it is done at every node the packet visits. The RFR method, however, does not have to embed the packet direction in the packet header which in turn save some redundant bits and the mechanism of checking them. Since only the complexity inside the node is considered, we conclude that the complexity of the RFR is approximately the same as the SCP which is O(D).

The NSC algorithm checks the addresses of the neighbors instead of itself. Because the number of neighbors is equal to 2Δ, the steps needed to do the comparison are 2Δ times D. That gives us the complexity of $O(\Delta \cdot D)$.

The PMC algorithm employs a different approach. It needs to match the largest common pattern between the neighbors of the present node and the destination addresses. Since the number of patterns that can be extracted from an address of length D is

$\frac{D(D+1)}{2}$, the required step counts is $\frac{D(D+1)}{2}$ times Δ. That gives us the complexity of the PMC routing method which is $O(\Delta \cdot D^2)$.

It is obvious that the complexities of the RFR and NSC are very close to that of the SCP algorithm, especially when Δ is small. In a network with a fairly large diameter, PMC may become more complex than the others. This situation is also reflected when running the emulation programs. The running times of SCP and RFR are nearly the same. The one of NSC is slightly longer and the one of PMC is the longest. When emulating a large network with more than one thousand nodes, there is significant time difference between the PMC and the others.

Combining the observations made previously, we can conclude that the RFR and NSC have certain advantage over the SCP method. Under nearly the same complexity, they achieve lower delay and high network throughput. The PMC method is good when the length of the addresses is small or the propagation delay along each link is large.

2.6 Summary

In this chapter, the principle of constructing the UdBG and the BdBG networks is introduced. Three new routing algorithms, namely Refreshing, Neighbor Searching and Pattern Matching algorithms, that can be used in the BdBG networks are proposed and their effects on performance are studied. The existing method, Shortest Consistent Path algorithm, is also included for comparison purpose.

Results in this chapter also show that the throughput of the BdBG network is more than doubled compared to the UdBG network. That justifies the efficiency of the bidirectional networks.

Three proposed routing algorithms have different effects on the performance of the BdBG networks. Generally, the PMC can achieve lower mean path length and consequently has lower delay in the low offered load region while the RFR and the NSC can

support higher throughput. From the study results including the complexity consideration, we conclude that PMC method can be used when the propagation delay along each link is rather large and the length of the node addresses is small; otherwise, the RFR and the NSC methods are more attractive due to their low complexities and more compromising performance in terms of delay and throughput.

Chapter 3

Hierarchical Structures of the UdBG and BdBG Networks

3.1 Introduction

It has been shown in the previous chapter that dBG networks can support a large number of nodes with considerable small mean path lengths. Therefore, they are certainly alternate choices for MAN structures when compared to other kinds of topologies for multihop systems. However, a network covering a wider area requires many cross-connections among the nodes. This may be an undesirable factor for MAN implementation as one may need to install many fibers connecting from one end to the other. One solution is to make the network hierarchical. Hierarchical structures can reduce a large amount of link cost and also are more adaptable to future network growth. Furthermore, hierarchical structures can make use of the locality in communication among adjacent nodes and therefore improve the network performance.

In addition to the parameters used before, the effects of the locality as well as the propagation time on the hierarchical structures will also be studied in this chapter.

3.2 Constructing hierarchical UdBG and BdBG networks

To build a hierarchical UdBG or BdBG network, we can first divide a large network into small clusters with each cluster forming a complete (Δ_L, D_L) UdBG or BdBG network by

itself, where the subscript L indicates local clusters. Then a particular node is chosen from each cluster as a *bridge node* that will connect to others in different clusters. The method of choosing a bridge node will be discussed shortly. Thus, all the bridge nodes form another (Δ_U,D_U) UdBG or BdBG network at the upper level which is indicated by the subscript U. We shall use the notation $(\Delta_L,D_L)/(\Delta_U,D_U)$ to indicate a general two-level network. For example, in Figure 3.1, local clusters are (2,3) BdBG networks while all the nodes with address 100 are chosen as bridge nodes form the upper level (2,2) BdBG network.

Let N be the number of nodes in the network and N_L be the number of nodes in one cluster. Therefore, $N = N_L \bullet \Delta_U{}^{D_U} = \Delta_L{}^{D_L} \bullet \Delta_U{}^{D_U}$. In this thesis, we only consider the case in which $\Delta_L = \Delta_U = \Delta$. Thus, the above equation is simplified to $N = \Delta^{D_L+D_U}$.

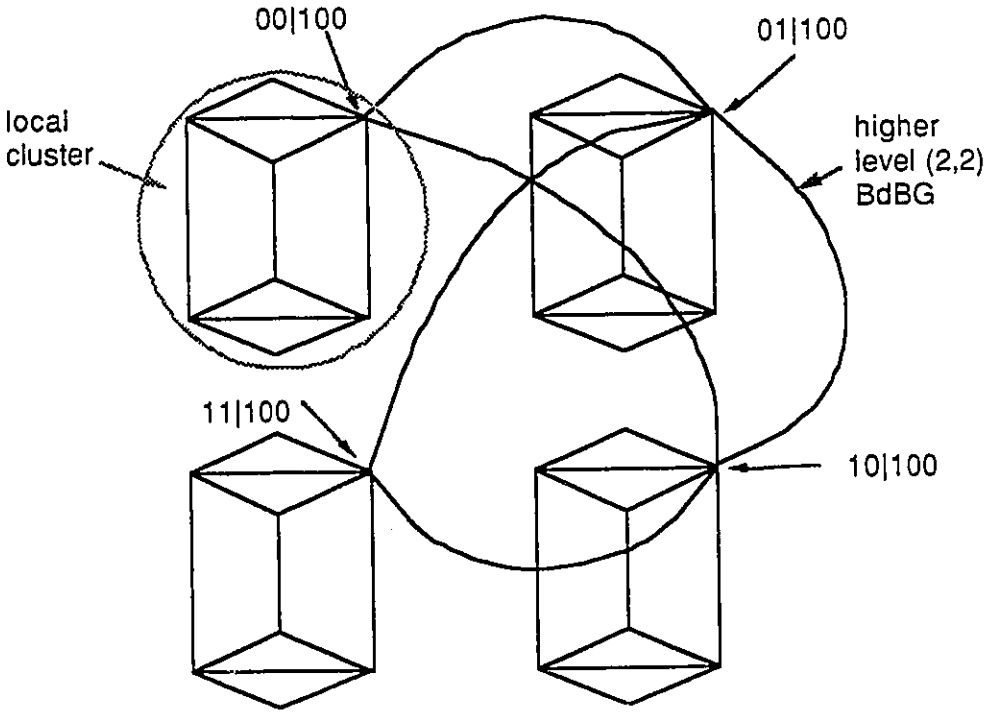


Fig. 3.1: Two level BdBG network in which low level is a (2,3) BdBG and upper level is a (2,2) BdBG. Nodes XX|100 are bridges.

In the two-level hierarchical BdBG networks shown in Figure 3.1, every node is represented by a unique address that contains two parts: the first part indicates the upper level BdBG address which also indicates the address of this particular cluster; the second

part indicates the node address inside this local cluster. For example, a node with address "00|100" indicates it is located in the cluster with address 00, and the local node address is 100. By this way, a node can distinguish local traffic by just checking the first part of the destination address. If the first part of the destination address does not match the address of the local cluster, the packet will be routed to the bridge node; otherwise, the packet will be directed to the local node with the address in the second part. For example, if node 00|001 in Figure 3.1 receives a packet with the destination address 10|111, it will first direct this packet to the bridge at node 00|100 which will send this packet to the cluster with address 10.

The other types of hierarchical networks can be formed similarly using this addressing scheme.

The method of choosing an appropriate bridge node is discussed in the following.

(1) Choosing a bridge node in UdBG networks

Define the node-mean-path-length, $\overline{H}$(node-i), as the average number of hops between node i and all other nodes in the network. Mathematically, $\overline{H}$(node-i) is given by

$$\overline{H}(\text{node - i}) = \frac{\sum_{j \in \text{nodes in network}} (\text{number of hops from node i to node j})}{N} + \frac{\sum_{j \in \text{nodes in network}} (\text{number of hops from node j to node i})}{N} \qquad (3.1)$$

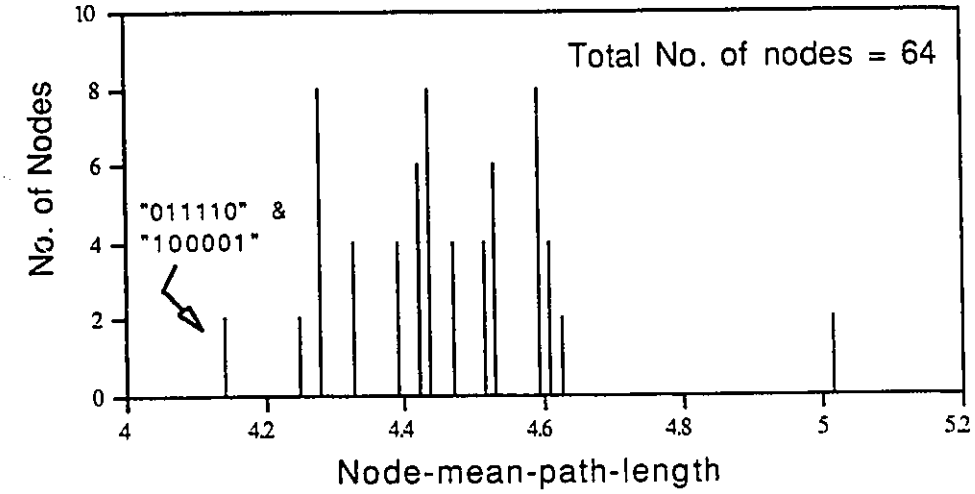
Note that the two path lengths are shown explicitly in the numerator because they are not necessarily equal. Unlike the other symmetric networks such as ShuffleNets and Manhattan Street Networks, $\overline{H}$(node-i) varies according to the location of node-i, and this obviously will affect the choice of the bridge location. In the following, mean path length of the bridge node will be expressed by $\overline{B} = \overline{H}(node - i)$, where node-i is the chosen bridge node.

As shown above, $\overline{H}$(node-i) differs depending on the location of node-i. Therefore, it is necessary to choose a node as the bridge node so that the mean path length of this node is minimum.

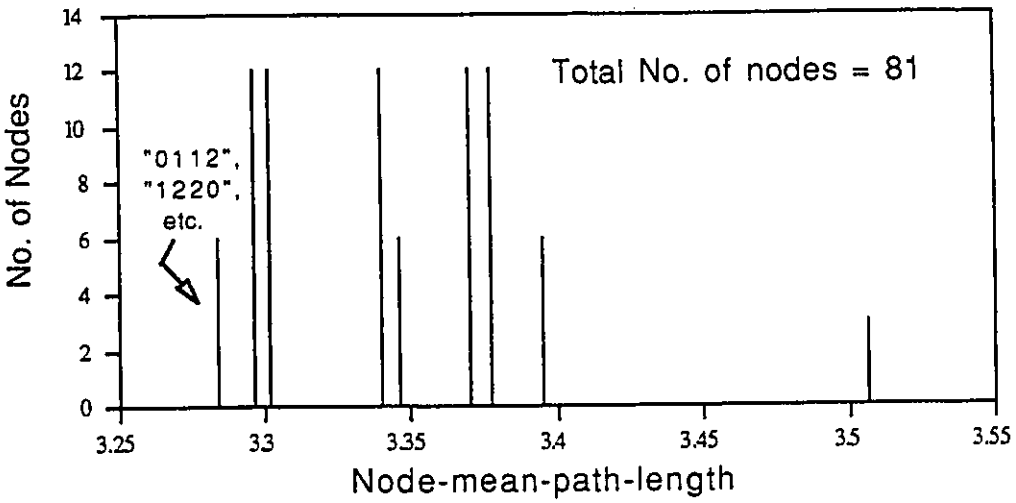
Computer programs written in C programming language have been coded to deal with this issue. Details of programs can be seen in Appendix B. Running times of the programs depend on the size of the network and the routing algorithms applied. A typical value would be about ten minutes running in DEC terminals for a network with one thousand nodes.

By the means of computer programs, each node can be searched and therefore one can obtain its node-mean-path-length. Figure 3.2(a) shows the distribution of the number of nodes with respect to the node-mean-path-lengths for a (2,6) UdBG network. In a total of 64 nodes, it is observed that $\overline{H}(011110)=\overline{H}(100001)=4.141$ which are the smallest node-mean-path-lengths in the network. Either one of these two nodes can be considered as the bridge node. Similarly, it is found that in all binary UdBG networks, the nodes with address of all 0's or 1's, except the first and the last digits, always have the shortest node-mean-path-lengths.

We have also investigated the UdBG networks when Δ is larger than or equal to 3. Figure 3.2(b) shows the distribution of node-mean-path-length in a (3,4) UdBG network. There are obviously 6 nodes which have the same minimum mean path lengths. In detail, they are "0112", "0221", "1002", "1220", "2001" and "2110". It is observed that in all UdBG networks when $\Delta \geq 3$, nodes with address pattern "xyy...yz" always have the minimum node-mean-path lengths, where x, y, z are integers belong to the set $(0,1,\ldots,\Delta-1)$ and $x \neq y \neq z$. This gives us a simple guideline to choose an optimum bridge node in the sense of minimum mean path length when constructing the hierarchical UdBG networks.



(a): A (2,6) UdBG network.



(b): A (3,4) UdBG network.

Fig. 3.2: Distribution of node-mean-path-lengths of UdBG networks.

(2) Choosing a bridge node in BdBG networks

Unlike the UdBG networks discussed above, the address-bits of the optimum node for the bridge does not follow a general pattern. For example, in a (2,4) BdBG network using RFR routing algorithm, node 0011 is an optimum choice; however, in a (2,5) BdBG network, node 01110 is found to be the best choice. More investigations using computer programs have shown that the best bridge location depends on the routing algorithms as well. Although there is no specific rule to pick the desirable bridge locations in a hierarchical BdBG network, our experience does suggest that, if the network is laid out on a plane in a symmetric way like the (2,4) BdBG network in Figure 2.1, the nodes closer to the center are more likely to have smaller node-mean-path length. So as a rule of thumb, they can be chosen as bridges. So when constructing a general network, one may decide first on the routing algorithm that gives an acceptable delay-throughput performance (e.g., RFR in the binary network). Then one would use the above method to choose a bridge location. More details in design issues will be discussed in Chapter 4.

3.3 Performance Analysis

For the same reason stated in Chapter 1, we shall use again the same approaches to carry out the performance evaluations of the hierarchical networks. Therefore, our measures will still be the mean path length, the delay and the throughput of the network. However, for the hierarchical networks, the method of deriving these measures would be different as discussed in this section.

3.3.1 Mean Path Length

To derive the mean path length of the network, an additional parameter, the locality factor α, which is defined to be the probability that both source and destination nodes of a message are in the same cluster, is introduced in this Chapter. This is a useful measure because although it is hard in general to predict the value of α in real network operation, a large

number of applications can be characterized by a communication structure that results in significant locality[DAND90]. Therefore, by varying α from 0 to 1, we can observe the network performance under different conditions.

Under the uniform traffic assumption declared in Chapter 2.4.2, we can assume that if each node sends a packet to every other nodes in the network, the total offered load to the network is N(N-1), where N is the number of nodes in the network. Therefore, based on this, the internal traffic can be calculated and so can the mean path length of the network which is the ratio of the internal and external traffic. Although the real offered load of the network may not be N(N-1), the change in the internal traffic accordingly will result in the same mean path length since the difference will be canceled out in the ratio. The mean path length of 2-level hierarchical networks can be derived as follows assuming all clusters are identical. Let

N = total number of nodes in the network;
m = number of clusters;
$\overline{H_1}$ = mean path length of the local cluster;
$\overline{H_2}$ = mean path length of the upper level network (bridges);
$\overline{H}$ = mean path length of the network;
$\overline{B}$ = mean path length of the bridge node;
α = locality factor.

Then,

$$\overline{H} = \frac{\sum \text{internal traffic}}{\sum \text{external traffic}} = \frac{m\sum \text{internal traffic in one cluster} + \sum \text{internal traffic in second level}}{N(N-1)} \tag{3.2}$$

The two terms in the numerator can be obtained,

$$\begin{aligned} &\sum \text{internal traffic in second level} \\ &= \overline{H_2} \cdot \sum \text{external traffic offered to the second level} \\ &= \overline{H_2} \cdot (1-\alpha) \cdot N \cdot (N-1) \end{aligned} \tag{3.3}$$

$$
\begin{aligned}
&\sum \text{internal traffic in one cluster} \\
&= \sum \text{local traffic} + \sum \text{remote traffic} \\
&= \overline{H_1} \cdot \alpha \cdot \sum \text{external traffic to one cluster} \\
&\quad + 2[\overline{B} \cdot (1-\alpha) \cdot \sum \text{external traffic to one cluster}] \\
&= \overline{H_1} \cdot \alpha \cdot \frac{N}{m} \cdot (N-1) + 2 \cdot \overline{B} \cdot (1-\alpha) \cdot \frac{N}{m}(N-1) \quad .
\end{aligned}
\tag{3.4}
$$

The reason for the factor 2 is that a message would go through its own cluster and the other cluster where the destination node is; therefore, two bridger-mean-path-lengths have to be considered.

Substitute (3.2) and (3.3) to (3.2), we get

$$\overline{H} = \alpha \overline{H_1} + 2(1-\alpha)\overline{B} + (1-\alpha)\overline{H_2}, \tag{3.5}$$

Equation (3.5) will be used to calculate the mean path length of the hierarchical networks throughout this thesis.

If the traffic of the network is assumed to be uniformly distributed, i.e., $\alpha = \frac{N/m-1}{N-1}$, Equation (3.5) can be revised as

$$\overline{H} = \frac{(N-m)\overline{H_1} + N(m-1)(2\overline{B} + \overline{H_2})}{m(N-1)} \tag{3.6}$$

Notice that we can always choose a bridge such that $\overline{B}$ is less than $\overline{H_1}$. When $\overline{B}$ is equal to $\overline{H_1}$, we can simplify (3.6) to

$$\overline{H} = \overline{H_1} + \frac{N(m-1)}{m(N-1)} \cdot (\overline{H_1} + \overline{H_2}) \tag{3.7}$$

It is observed that if $m >> 1$ and $N >> 1$, Equation (3.7) can further be simplified to

$$\overline{H} \approx 2\overline{H_1} + \overline{H_2}. \tag{3.8}$$

The result is obvious because in an average sense, each message would go through its own cluster, the upper level network and the other cluster. That results in the total mean

path length of the network which is the sum of the mean path lengths of two local clusters and one upper level network.

3.3.2 Average Packet Delay

The average packet delay of the network is evaluated using Equation (2.2) which is restated here for the sake of convenience.

$$\overline{D} = \frac{1}{N\ (N-1)} \sum_{i=1}^{J} \frac{L_i}{\mu C - L_i \lambda} + \overline{H} \cdot \tau \ . \qquad (3.9)$$

However, since the propagation time plays an important role when the network is large, especially when the inter-cluster propagation time is rather larger than the intra-cluster one which is common in the hierarchical structures, it is necessary to analyze its effect on the performance of the hierarchical networks. Therefore, in Equation (3,9), the propagation time at each link τ can no longer be assumed to be equal at every link in the network. In contrary, it is divided into two parts. Assume the traffic is uniformly distributed. Let τ_L be the propagation time at each link in a local cluster(intra-cluster), τ_U be the one in the upper level(inter-cluster). Hence, the propagation delay part of Equation (3.9) should be considered as the summation of the propagation delay in the lower and the upper level. Consequently, a weighted sum of the propagation delay expressed as $\overline{H_1} \cdot \tau_L + \frac{N(m-1)}{m(N-1)} \cdot (\overline{H_1} + \overline{H_2}) \cdot \tau_U$ is used in the analysis instead of just $\overline{H} \cdot \tau$ in Equation (3.9) under the uniform traffic assumption.

In a one-level network used as comparison in this chapter, we assume the propagation time at each link is equal to the one in the upper level of the hierarchical networks, i.e. $\tau=\tau_U$. Furthermore, we define the ratio $r = \tau_U / \tau_L$ to be the proportion of two propagation times. We shall discuss various situations with different r.

3.4 Performance of Hierarchical dBG Networks

3.4.1 Uniform Traffic Case

Assume the traffic is uniformly distributed and the propagation time at each link is neglected. Note that these two assumptions would give the worst performance of the hierarchical structures when compared to the one-level network because the original intention of the hierarchical structures is to make use of the locality and the inter-cluster propagation effect. However, even under such assumptions, if the network performance is better than the one-level networks, that would justify the usage of the hierarchical networks.

(1) UdBG Networks

A 1024-node binary network is considered, and the cluster size N_L is varied from 4 to 256 nodes. Under the uniform traffic condition, the locality factor is $\alpha = \frac{N_L - 1}{1023}$, and it ranges from 0.00293 to 0.249 which in most cases are small except for the (2,8)/(2,2) network. Their performance in terms of the mean path length is tabulated in Table 3.1. In general, networks with a large number of clusters have better performance. However, the one-level UdBG network has the shortest mean path length than any other UdBG hierarchical networks. This suggests that the one-level network is adequate in the design of UdBG networks that have uniform traffic.

Table 3.1: Results of mean path lengths of two-level hierarchical dBG networks with 1024 nodes in uniform traffic case.

1_{st} level/2_{nd} level	No. of nodes per cluster	No. of clusters	mean path length	
			UdBG	BdBG
(2,2) / (2,8)	4	256	8.397	7.097
(2,3) / (2,7)	8	128	8.665	7.148
(2,4) / (2,6)	16	64	9.188	7.596
(2,5) / (2,5)	32	32	9.892	8.438
(2,6) / (2,4)	64	16	10.680	9.479
(2,7) / (2,3)	128	8	11.405	10.478
(2,8) / (2,2)	256	4	11.793	11.138
One-level (2,10) network			8.377	7.547

(2) BdBG Networks

The mean path lengths of the two-level binary BdBG networks with 1024 nodes have been carried out under the same assumption made in the previous paragraph. Results are also shown in Table 3.1. The RFR routing algorithm is used in the BdBG networks. Unless otherwise specified, the routing algorithm used in the BdBG networks is assumed to be RFR hereafter since our analysis in Chapter 2 indicated that it gives the best compromise between delay and throughput performance.

Unlike the UdBG networks in which the one-level network has the shortest mean path length, Table 3.1 shows that the (2,2)/(2,8) and (2,3)/(2,7) combinations have even shorter mean path lengths than the one-level network. It suggests that by simply constructing the hierarchical BdBG networks, we can have the achievement in terms of the mean path length over the regular one-level network. We should expect that the hierarchical networks have even better performance if propagation time or locality is considered because the assumptions we made give the worst performance of the hierarchical networks.

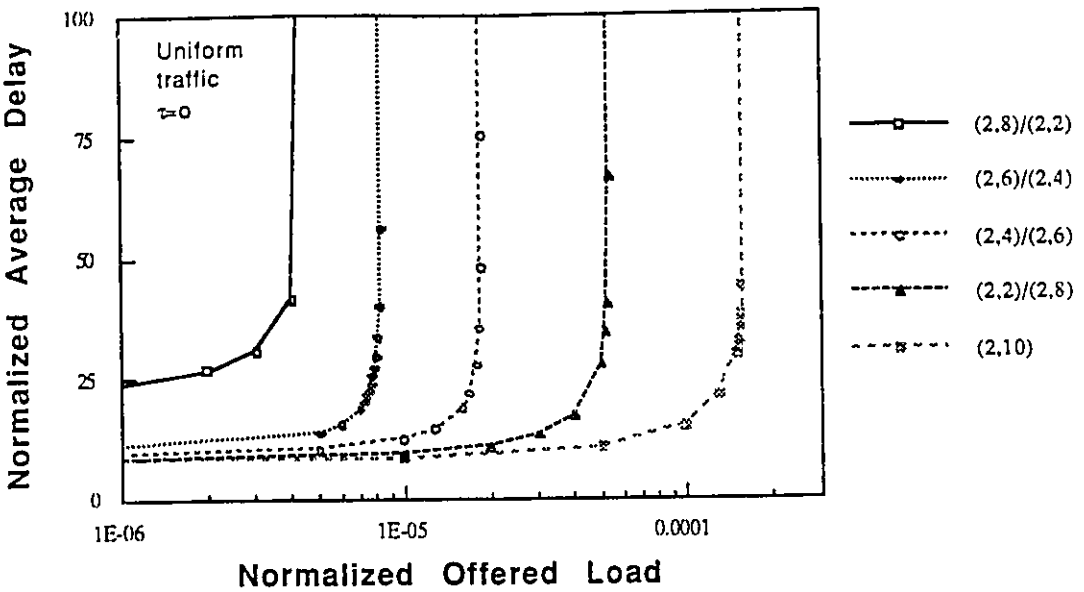


Fig. 3.3: Comparison of 2-level binary UdBG networks with 1024 nodes.

The delay and throughput performance of the same 2-level binary UdBG network is shown in Figure 3.3 assuming the traffic is uniform and the propagation time is zero at each link. Note that under this assumption, the one-level UdBG network has the best performance in both delay and throughput. As the number of clusters increases, the throughput of the network also increases because the edge load in the second level has become less.

The delay and throughput performance of the two-level BdBG network under the same assumption has also been investigated. Figure 3.4 is one such example for showing the performance of 2-level binary BdBG networks with 1024 nodes. SCP routing algorithm is used. We see that the (2,2)/(2,8) combination again has the best performance in terms of delay and throughput among the hierarchical structures as in the UdBG networks.

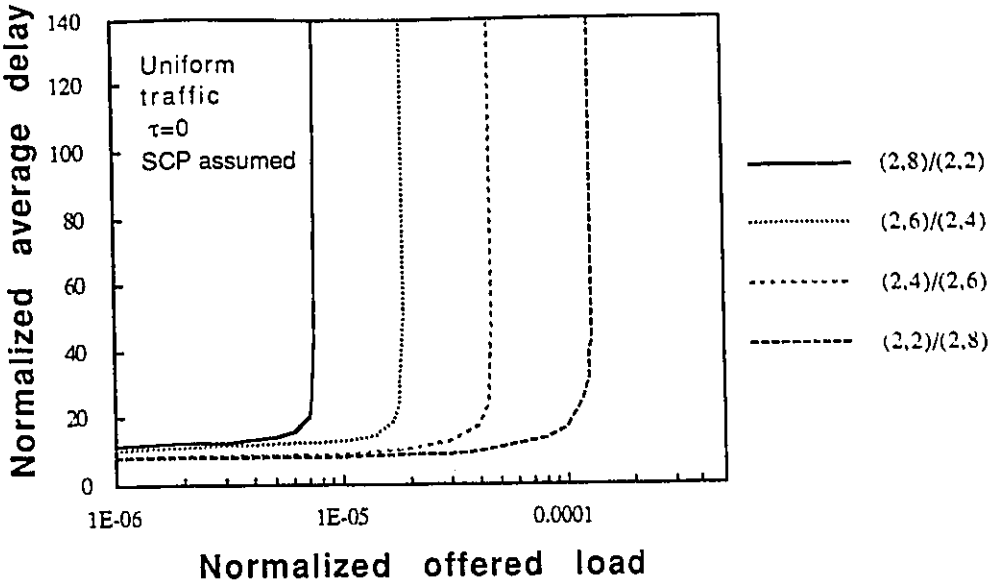


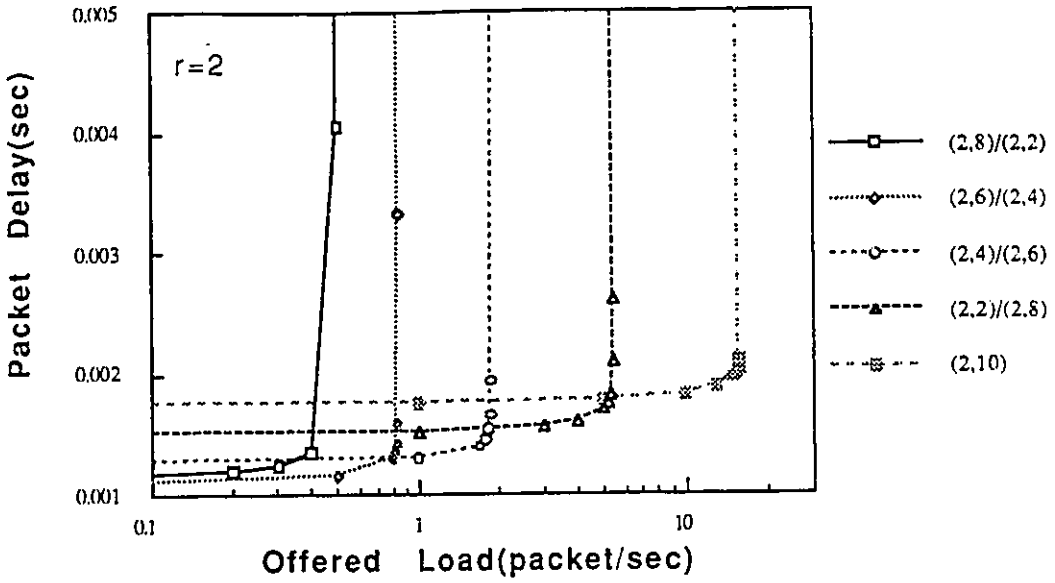
Fig. 3.4: Comparison of 2-level binary BdBG networks(SCP assumed).

3.4.2 Propagation Effects

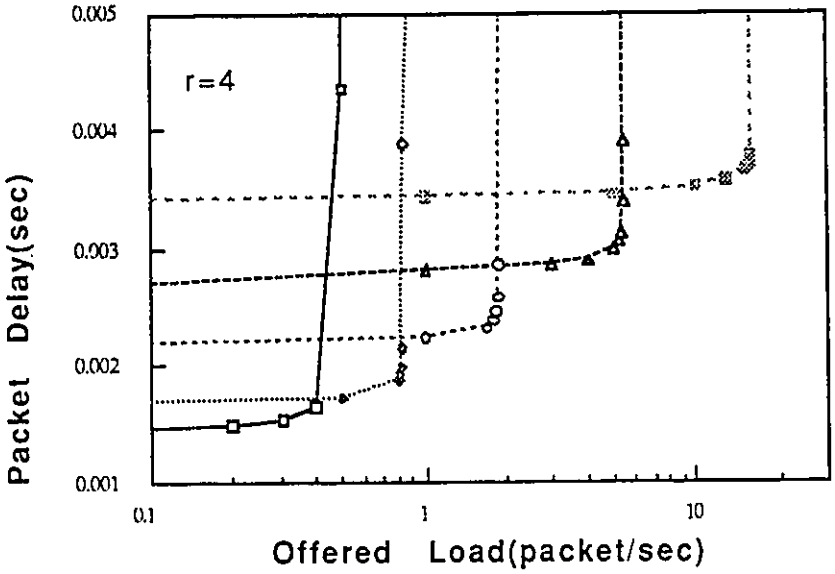
Using the same method as in Section 3.3.2, the results of the propagation effect on the hierarchical networks are shown in this section.

(1) UdBG Networks

In Figures 3.5(a) and 3.5(b), the analysis results of a 1024-node binary UdBG networks with the same parameter values in the previous section are presented. Assume the intra-cluster propagation time τ_L is 100μs and the inter-cluster propagation time $\tau_U=r\cdot\tau_L$, where r takes on different values. Furthermore, assume $1/\mu$=10,000 bits, C=1Gb/s. These two diagrams show the network delay and throughput performance when r equals to 2 and 4.



(a): r=2



(b): r=4

Fig.3.5: Delay and throughput performance of hierarchical UdBG networks.

Compared to Figure 3.3 in which the one-level network has the best performance in terms of both delay and throughput, it shows that in Figure 3.5(a) and 3.5(b), the one-level network has the highest delay in the low offered load region since the propagation time is the dominant factor in this situation. It is also observed that as the ratio r increases, the average packet delay increases as expected. This is because r measures indirectly the distance between two clusters. However, in the one-level network, the delay increases more rapidly than the one in the hierarchical networks. For example, when r=2, the difference of delay between the one-level (2,10) network and the (2,8)/(2,2) network is about 0.67ms in the low offered load region. On the other hand, this difference becomes 1.95ms when r=4. As a general observation from other evaluations, if the inter-cluster distance is much larger than the intra-cluster distance, then the hierarchical structures would certainly have more advantages. This is an important consideration for network design.

An interesting point is that Figure 3.5(a) shows that the delay of (2,8)/(2,2) is larger than the one of (2,6)/(2,4) in the low offered load region when r=2; however, Figure 3.5(b) shows the reverse when r=4. That is because the propagation factor effects more when more inter-cluster traffic happens.

On the other hand, as we observe in Figures 3.5(a) and 3.5(b), the network throughput does not change when considering different propagation factors. That is due to the fact that the maximum edge loading in the upper level is the dominant factor for the network throughput. This can also be seen from the first part of Equation (3.9) which goes to infinite when the normalized offered load $\lambda/\mu C$ approaches the maximum edge loading L_i. We shall see that in the next section the throughput performance would be improved if we consider another important element in the hierarchical structures; i.e. the locality factor.

(2) BdBG Networks

Assume the intra-cluster propagation time on each link is τ_L=50μs, the inter-cluster propagation time is $\tau_U=r\cdot\tau_L$, where r takes on different values. Furthermore, we use the same parameters as in the UdBG networks, i.e. 1/μ=10,000 bits, C=1Gb/s.

Analysis results are shown in Figure 3.6. An example of (2,2)/(2,8) BdBG network compared to one-level (2,10) BdBG network is given. Three sets of curves show the results for r equal to 2, 4 and 10 respectively. That corresponds to the inter-cluster propagation time of 100, 200 and 500μs respectively. It is observed that in the low offered load region, the packet delays of the (2,2)/(2,8) BdBG network are lower than the ones of the one-level (2,10) BdBG network. As the ratio r increases, the gap between the two curves in one set is becoming larger. For example, when r=2, the difference of the delay between the hierarchical and the one-level network is 0.133ms in the low offered load region; on the other hand, when r=10, it becomes 0.89ms. Notice that the (2,2)/(2,8) network has more clusters than other combinations. As shown before in Figures 3.5(a) and 3.5(b), when the number of clusters becomes larger, the propagation time has more impact on the packet delay. Therefore, we expect that other combinations, such as (2,2)/(2,8), etc., have much lower delay than the one-level network.

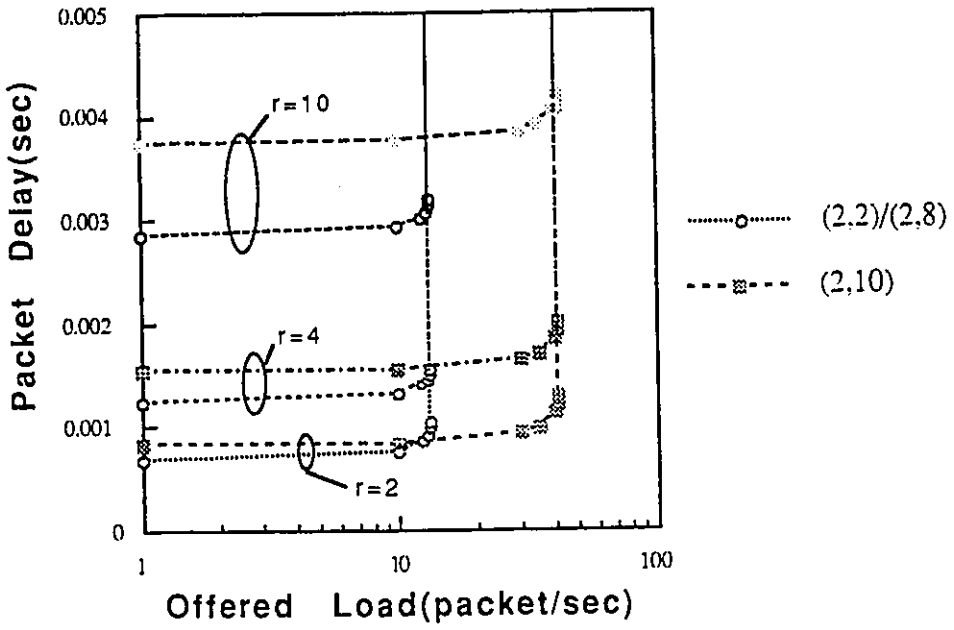


Fig. 3.6: Comparison between the (2,2)/(2,8) network and the one-level (2,10) network with different ratio r.

It is also observed that the propagation time does not change the network throughput due to the same reason stated previously. We shall see the network throughput is effected by the locality factor α.

3.4.3 Locality Factor

To study the effect of the locality factor α on the network performance, we fix the propagation delay τ=0 on each link so that we can illustrate its effect on the performance of the mean path length and the normalized packet delay.

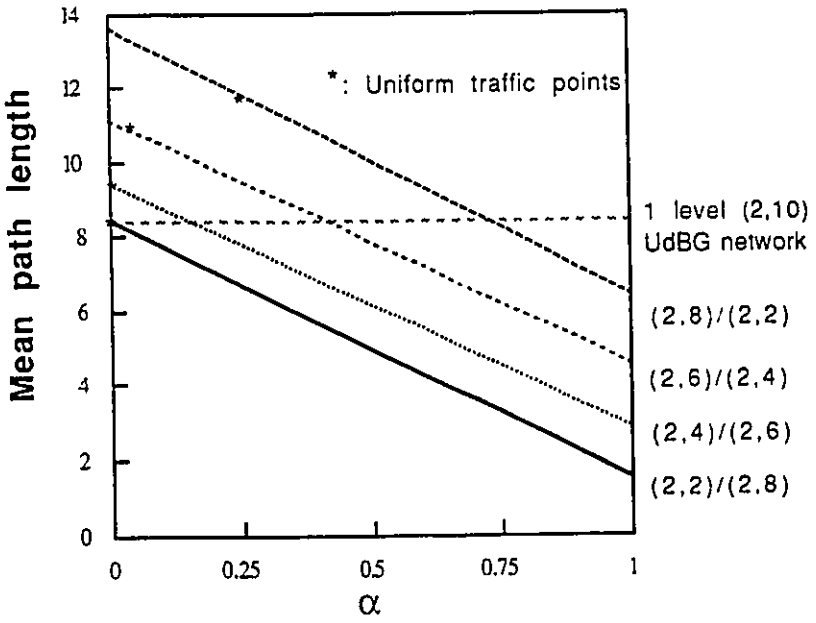


Fig. 3.7: Mean path length vs locality factor α in UdBG networks.

(1) UdBG Networks

Equation (3.5) is used to evaluate a 1024-node binary UdBG network with different combinations. Figure 3.7 shows explicitly the mean path length performance as a function of the locality factor α. The relationship is linear as expected. The one-level (2,10) network is used as comparison. The localities of the uniform traffic cases (indicated by an

asterisk) are also included for reference. Our analysis results show that the hierarchical structures will eventually perform better if the α exceeds a certain value. For example, in the (2,4)/(2,6) network, when α exceeds 0.13, the mean path length of the network will be less than the one-level (2,10) network.

Figure 3.8 illustrates the comparison among different combinations of 2-level 1024-node hierarchical UdBG networks with α=0.5. It is observed that as the number of clusters increases, the network performance is getting better. Among all combinations, the (2,2)/(2,8) has the best performance.

Figure 3.9 shows the delay and throughput performance of the (2,4)/(2,6) UdBG network with different locality factor. It is observed that when α increases, the network throughput significantly increases. This is because the maximum edge load in the upper level is reduced as the locality increases. The average packet delay is also improved as shown in this figure. This again justifies the advantage of the hierarchical structures.

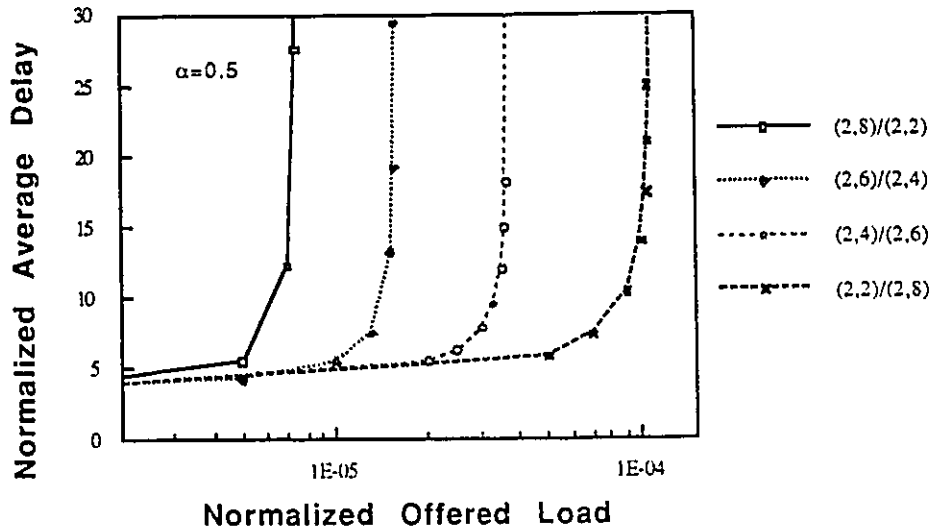


Fig. 3.8: Performance of 1024-node UdBG networks with different combinations.

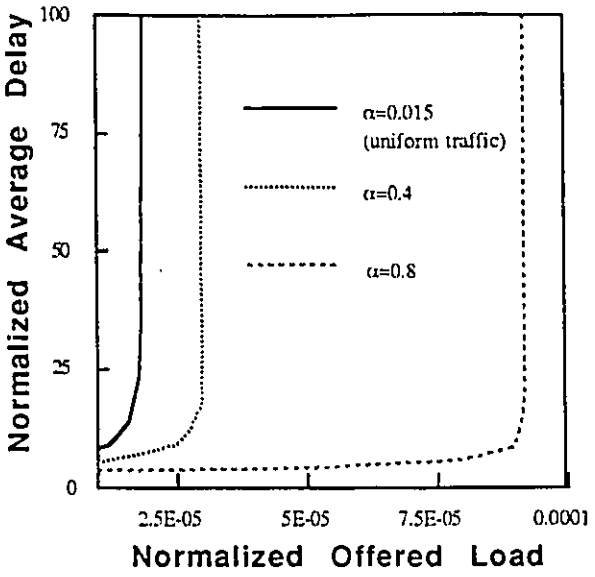


Fig.3.9: Comparison among different locality factors in a (2,4)/(2,6) UdBG network.

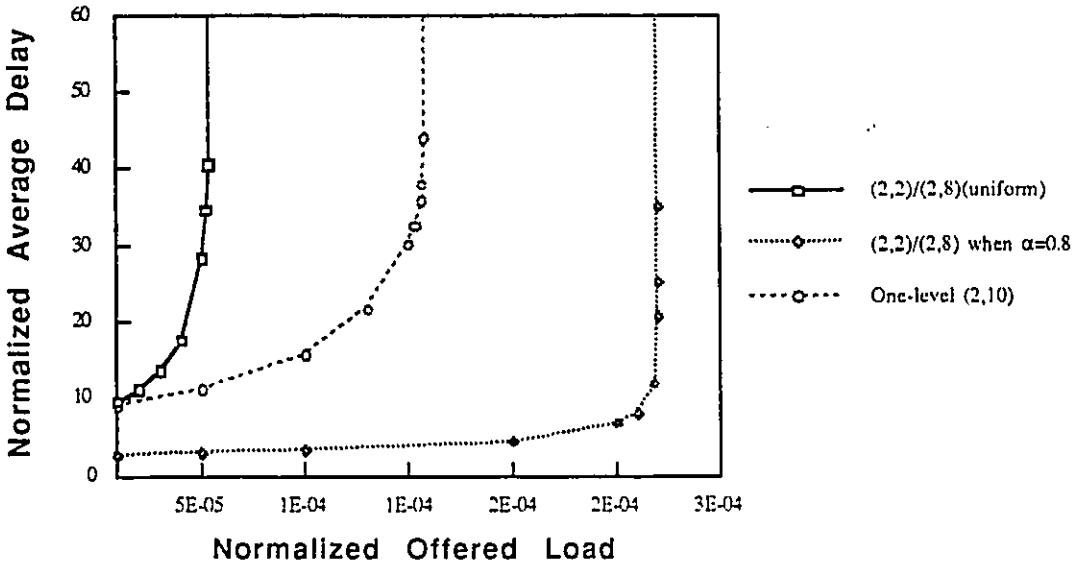


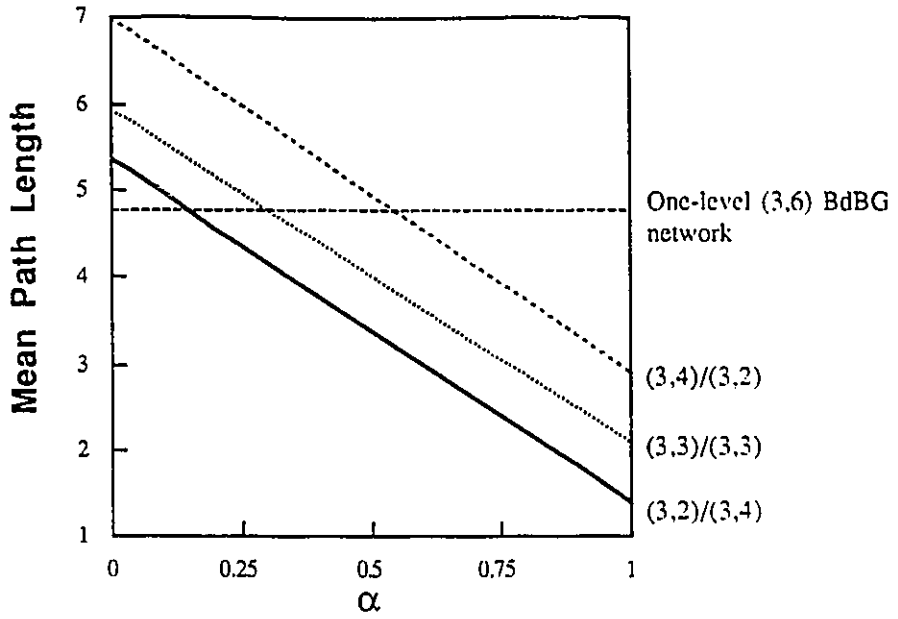
Fig.3.10: Comparison between a (2,2)/(2,8) and (2,10) UdBG networks.

Figure 3.10 compares the delay performance of the one-level network and the (2,2)/(2,8) hierarchical network under different locality factors 0.8 and 0.0029 which indicates the uniform traffic situation. Throughput of the hierarchical network is much higher when α is considerably large. It is observed that the delay curve of the one-level network is in between the two curves of the hierarchical network with α equal to 0.0029 and 0.8 respectively. It suggests that when α exceeds a certain point, both delay and throughput of the hierarchical networks would be better compared to the one-level network.

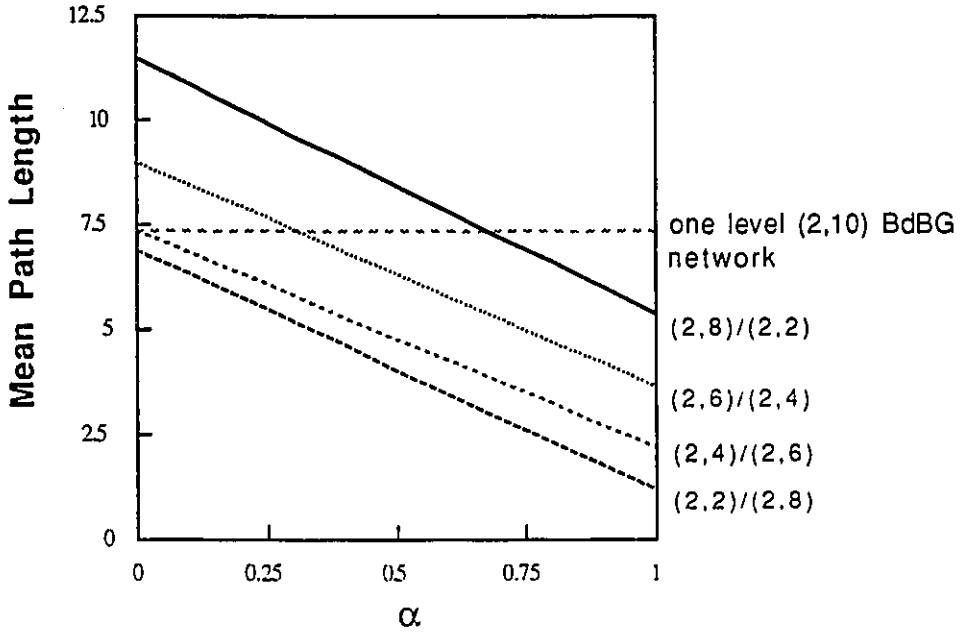
(2) BdBG Networks

Under the same assumption as in UdBG networks, analysis results of the hierarchical BdBG networks are shown below.

Figures 3.11(a) and 3.11(b) demonstrates the network performance in terms of the mean path length as a function of α for different hierarchical BdBG networks. Figure 3.11(a) shows a 729-node network based on node degree of 3 where the cluster size varies from 9 to 81 nodes, while Figure 3.11(b) shows a 1024-node network based on node degree of 2 where cluster size varies from 4 to 256. Just like the UdBG networks, as the locality factor increases, the mean path lengths of the hierarchical networks decrease linearly and eventually become lower than the one level network. As in Figure 3.11(a), when α is small, the one-level (3,6) network has shortest mean path length and it remains so until α exceeds a certain value which depends on the different combinations. However, the interesting observation is that the entire line of the mean path length of the (2,2)/(2,8) network in Figure 3.11(b) is under the one of the one-level (2,10) network. This means that even when there is no locality at all, i.e. α=0, the (2,2)/(2,8) network can still achieve a shorter mean path length than the one level counterpart. Thus this example demonstrates the advantage of using hierarchical structure in BdBG networks.



(a): Ternary hierarchical BdBG network with 729 nodes.



(b): Binary hierarchical BdBG network with 1024 nodes.

Fig. 3.11: Mean path lengths vs locality factor in hierarchical BdBG networks.

With the same locality factor α=0.5, Figure 3.12 shows the comparison among different combinations of 1024-node networks. The 1-level (2,10) network is also shown for comparison purpose. It is observed that when the number of clusters increases, the network performs better. Among all combinations of 2-level networks, the (2,2)/(2,8) has the best performance. Most of the network delay curves, except the (2,8)/(2,2) combination, are lower than the one of the 1-level network in the low offered load region. Unlike the uniform traffic case, the throughput of the (2,2/(2,8) network is even higher than the one of the 1-level network.

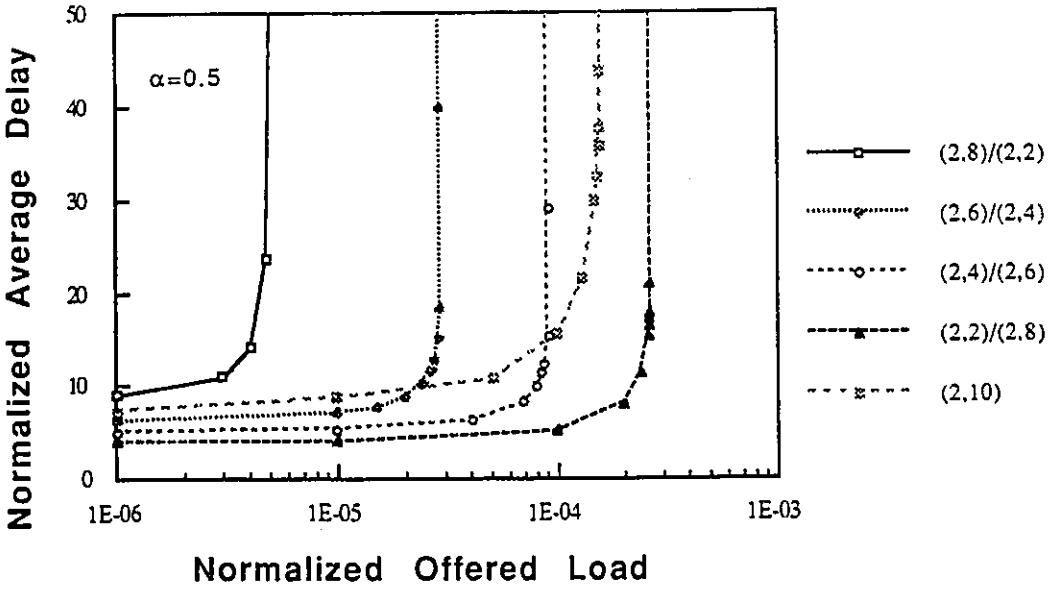


Fig. 3.12: Performance of 1024-node BdBG networks with different combinations.

Figure 3.13 gives the mean path length performance as a function of the number of clusters for various α. It shows clearly that when the number of clusters equals to 256(cluster size=4), the network has the optimum mean path length performance and this minimum is independent of α. Similar observations are made for the UdBG networks. This in turn explains the similar observation in Table 3.1 where uniform traffic is assumed.

In summary, when designing a metropolitan area network with 1024 nodes, the (2,2)/(2,8) combination is the best choice at least in terms of the network mean path length.

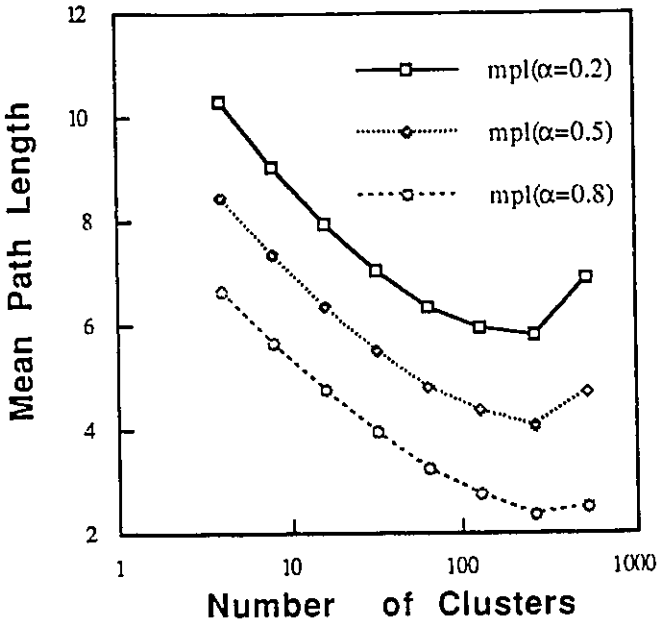


Fig.3.13: Network mean path lengths of a 1024-node BdBG network VS different number of clusters.

In Figure 3.14, we compare the hierarchical network with respect to the one-level network. The (2,2)/(2,8) and the (2,10) are chosen respectively. Two cases are shown with different values of locality factor α. We see that in the uniform traffic case where α=0.00293, the throughput of the network is lower than the one of the (2,10) network. However, when α=0.8, the hierarchical network throughput is well above the one-level network, and the delay of the hierarchical network is also improved as well. It shows that the hierarchical structures can take advantage of the locality factor so that the network can perform much better, especially in terms of the throughput performance.

Figure 3.15 compares the delay and throughput performance among the three routing algorithms. NSC is expected to have similar performance as RFR; therefore it is omitted here for clarity purpose. The (2,2)/(2,8) networks are used and the locality factors α =

0.8 is considered. It is shown that the RFR routing algorithm not only can achieve the lower delay of the network but also the higher throughput performance. Although the PMC has lower delay in the low offered load region, it has relatively lower throughput. By comparison, it is also observed that delay-throughput improvement of RFR is higher for a larger value of α.

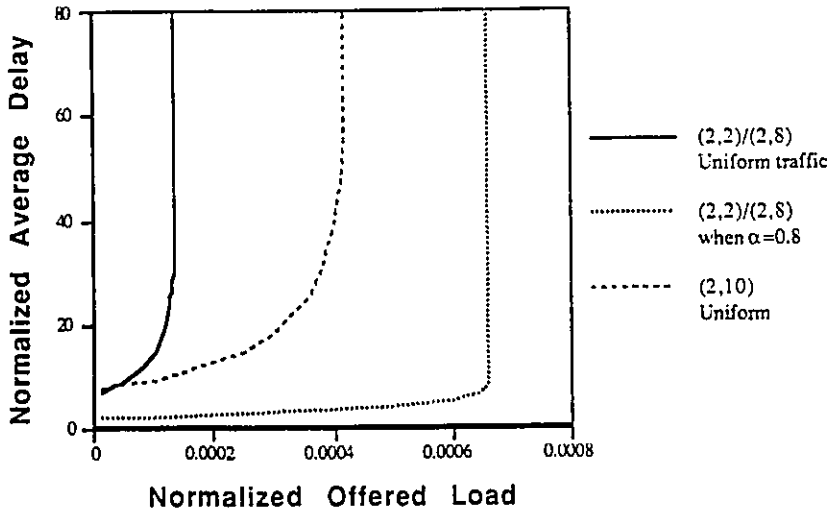


Fig.3.14: Comparison between (2,2)/(2,8) and one-level (2,10) BdBG network.

Experiments have also been performed by having different routing algorithms at different levels. In every combination, results show that the performance is very close to that when both levels are using the same routing method in the upper-level. For example, we can have RFR in the upper-level while SCP in the lower-level, and its performance is almost the same when both levels are using RFR; however, as the locality factor becomes larger, the routing in the lower-level may become more and more effective. Consequently, one should use methods that would achieve higher throughput in the upper level while keeping the routing method in the lower-level as simple as possible unless the lower-level becomes more congested than the upper level.

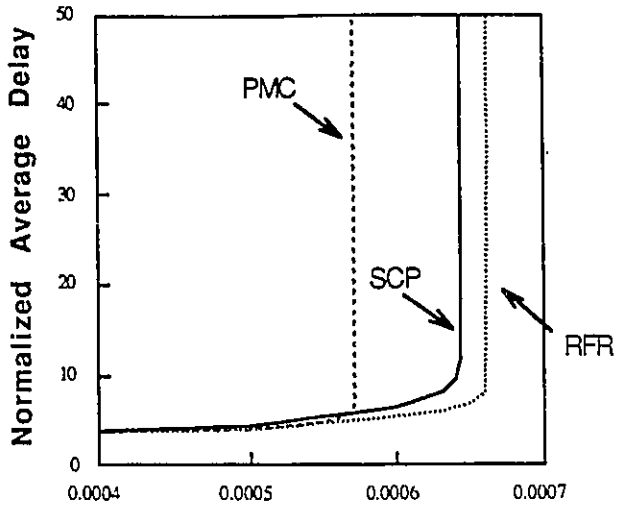


Fig.3.15: Comparison among SCP, RFR and PMC routing algorithms in a (2,2)/(2,8) BdBG network when α=0.8.

3.5 Summary

In this chapter, we introduce the method of constructing the hierarchical structures of UdBG and BdBG networks by dividing nodes into local clusters, choosing one bridge node at each cluster and then connecting them together.

The method of choosing an appropriate node as the bridge node is also studied. It showed that in UdBG network, some particular nodes with certain addressing pattern can be considered as the optimum choices. On the other hand, in BdBG networks, there is no such rule that can be found; therefore, one needs to use a computer program to derive the best bridge node location.

It has been shown that the hierarchical structures of UdBG and BdBG networks have better network performance than the one-level networks in terms of network mean path length, packet delay and throughput performance, especially when considering the propagation and the locality effects. It shows that the propagation time on each link affects

the packet delay in the low offered load region while the locality factor has effects on the throughput of the networks.

It also shows that when designing a 1024-node network using the binary scheme, the (2,2)/(2,8) is the best combination. Furthermore, in general, the RFR routing algorithms has been proved to be more promising in terms of the packet delay and network throughput.

Chapter 4

Optical Implementations and Design Criteria of dBG Networks

4.1 Introduction

Optical WDM multihop systems make use of multiple wavelengths to tap into the vast optical bandwidth. In general, the optical transmitter(s) and receiver(s) of each node in such systems are tuned to a fixed wavelength different from others and remain so from then on. Therefore, fast tunable transceivers that are not quite practical yet are avoided. When a node receives a packet, it just checks whether or not the packet belongs to itself and then decides to receive or choose an appropriate outgoing link to transmit the packet. This is done according to the routing algorithm which the system applies. The processes of routing and retransmitting in a node normally are done electronically in most proposed multihop systems since this is the original idea. As discussed before, the opto-electronic bottleneck caused by these processes becomes a major drawback of the multihop systems.

In this chapter, an optical implementation of nodes in the BdBG networks is proposed such that the data information in the packet remains in optical domain during the whole transmission. However, it still requires the headers of the packets to be converted and processed in the electronic domain in order to accomplish the routing control function. Nevertheless, this makes the node optically transparent to the data information. As the technologies of optical devices are being developed very quickly, once the optical routing parts are more mature and practical, we can actually replace the routing control boxes by the all-optical counterparts so that the network performance can be more enhanced.

The method of implementing the nodes of the BdBG networks is introduced in this chapter, including the one-level and the hierarchical networks. The general design criteria for the de Bruijn Graph networks are also raised. They aim to give designers the general steps to follow when constructing a MAN based on the dBG network.

4.2 Optical Components and Devices

The physical implementation of the node structures we propose is based on some available optical components and devices. For the reason of simplicity, some well-known basic optical components will not be introduced in detail in this section. For example, couplers(Fused biconical tapered couplers), that allow optical signals to split into two or more branches, can be found in Chapter 3 of [GREE93]. Instead, we shall introduce briefly some essential components and devices below.

4.2.1 Optical Switches and Directional Couplers

Optical switches, which normally consist of ON-OFF optical gates and branching optical circuits, have been developed rather advanced. For example, a high speed switch can be operated is less than 1ns[IKED83].

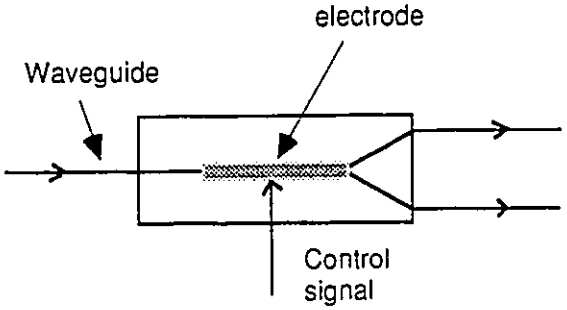


Fig. 4.1: A 1 X 2 directional coupler.

Among many 2x2 switches, the directional coupler switch is used most widely[OKYA91]. As shown in Figure 4.1, the 1x2 directional coupler is controlled by the

electronic bias voltage applied to the electrodes which are placed on the waveguide so that the optical signal can be directed to either two outgoing fibers.

Although optically controlled switches and directional couplers are available nowadays[CAVA90], we will consider only the electrically controlled switches and directional couplers in this thesis. This is because the proposed implementation design does not require all optical operations. Therefore, in order to make the system to be more efficient and cost-effective, this decision is made. Moreover, the technology has made electrically controlled switches and directional couplers very reliable. For example, the switching speed of less than 8ns[EISE92] can be easily achieved which is suitable for our proposed implementation in this chapter.

4.2.2 Optical-Fiber-Loop Memory

Because of the packet switching characteristic of the BdBG networks, each node needs packet buffering to solve the problem of contention. Using the optical-fiber-loop memory, buffering can be obtained by optical means[CALZ93]. A simplified diagram of the fiber-loop-memory proposed in this paper is shown in Figure 4.2.

In such a fiber-loop-memory, when a packet comes in, it enters the fiber loop through the 2x2 coupler. The circulation of the packet is controlled by electronic pulses that apply to the amplifier electrode to switch off gain. The switchable optical amplifier is used to compensate round-trip losses. The narrowband filter reduces the spontaneous noise generated by the amplifier and also allows wavelength's selectivity. In principle, the loop length could be equal to the packet duration; however, it is slightly longer than the packet duration to permit periodic switching-off of the amplifier. To release the packet, one can control the gate of the amplifier as well as the optical switch at the output link. Notice that the packet has to be released at the multiple time of one cycle of the fiber loop. The gate of the amplifier ensures to erase the packet in the loop. Therefore, with this device, one can store the packet in the optical domain.

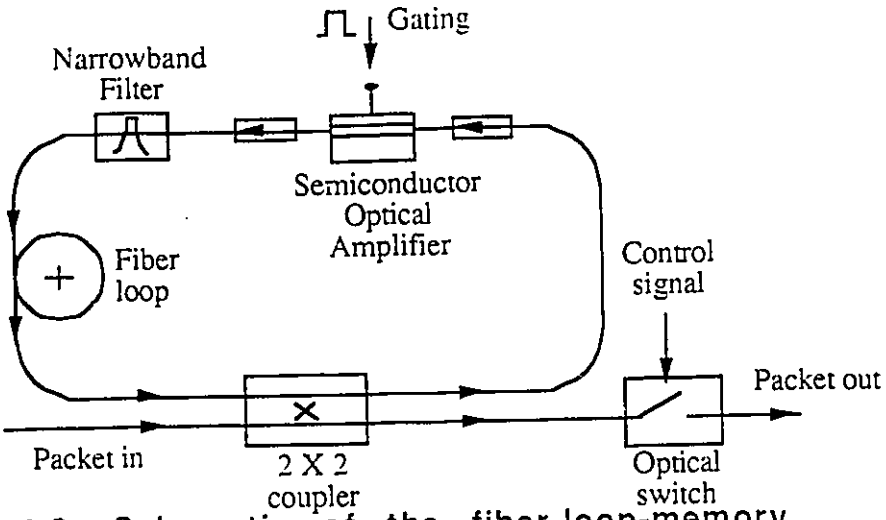


Fig.4.2: Schematic of the fiber-loop-memory.

4.3 Packet Scheme

To fit into the optical operation in a node, the packet format must accord with the demands. A general scheme of a packet for the BdBG networks is introduced below.

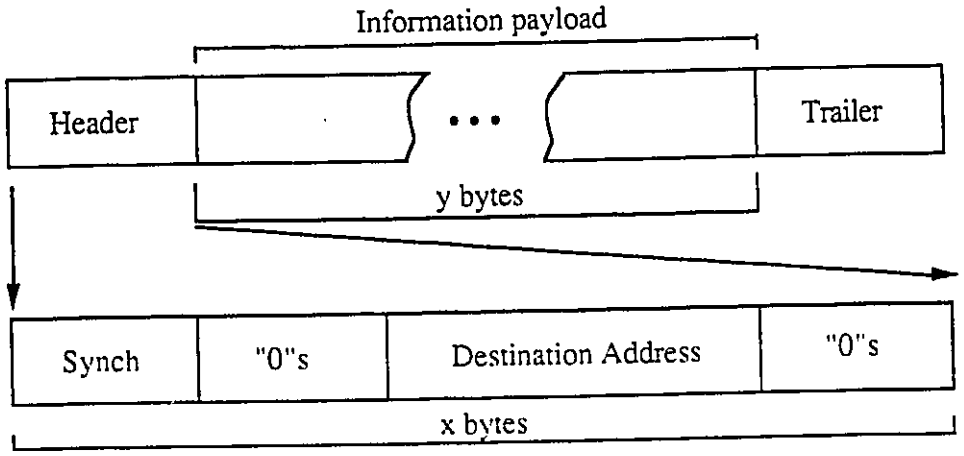


Fig. 4.3: Structure of the packet and its header.

Assume the packet length is fixed so that it can be manipulated more easily in the optical domain. In a general (Δ, D) BdBG network, each node address is represented by a

Δ-ary code. If Δ=2, i.e. binary network, the address is a binary code. In this case, the address can be put into the packet header directly in the bit form. However, if the address is not binary-coded, it has to be encoded to binary code at the transmitter side before it is transmitted in terms of bits, i.e. "0" or "1", and decoded at the receiver side.

Figure 4.3 shows the packet pattern and the header format. Assume the information payload has y bytes and the header has x bytes. These values may vary in different networks and mainly depend on the node addresses.

The first byte of the header which contains a special bit pattern designed for the synchronization purpose. So is the trailer of the packet which may contain in addition the error checking redundant bits. The second and the last bytes of the header are all "0"s representing the absence of the pulse. These two silent time slots ensure the operation of the optical devices as discussed later on.

The portion in the middle of the x-byte header represents the address of the destination node to which the packet belongs. For example, in a (2,4) one-level BdBG networks, only 4 bits are needed to indicate the packet destination address. In hierarchical networks, the node addresses contain two parts as discussed in Section 3.2. The first part indicates the cluster address and the second part is the node address in the local cluster. Therefore, the only thing that is different from the one-level networks is that it needs some extra bits to represent the first part. For example, in a (2,2)/(2,8) BdBG network, the address of a node may be 01|00110110. Thus, it needs 2 bits for the address "01" and 8 bits for the address "00110110". Considering the synchronization byte and the two silent slots, there are total 5 bytes in the header in this example.

4.4 Node Structure

In this section, we shall use the BdBG networks as examples to show the physical structure of the nodes. With the same scheme, networks with different configuration can also be taken into account. Note that the number of transceivers in each node depends on the de-

gree of the node. For example, in a Δ=2 BdBG network, most nodes would have 4 transmitters and receivers. Some nodes may have fewer number of transceivers as the discussion in Chapter 2.

Although not discussed, we assume that the coupling loss and the energy requirement for the optical devices can be compensated by necessary optical amplifiers wherever required.

4.4.1 One-level BdBG Networks

Figure 4.4 shows the physical structure of a node in a (Δ,D) one-level BdBG network. There are 2Δ incoming links and 2Δ outgoing links connected to the node. Each node in the network is divided into several functional blocks which correspond to the number of transceivers in the node. Each block represents an input and the possibility to transmit to one of the outputs, and it performs the operation of reception, routing and transmission. Only one functional block of the total 2Δ is illustrated in detail in Figure 4.4. The others are identical and omitted in the diagram. For example, in a node of (2,D) network, there will be 4 functional blocks inside the node.

When a packet arrives at the node, the photo-diode detects the arrival of the packet by recognizing the special pattern of the synchronization byte. Then it closes the optical switch for a period of time equal to the duration of the portion of the destination address in order to direct the address into the address control unit. The remaining of the packet continues on towards the routing control unit. The "0"s before and after the address part allow sufficient time for the photo-diode to operate. Assuming the network is operating at 1Gb/s, then 8 bits give the optical switch 8ns time to close. This is feasible nowadays such as for example, optical gates introduced in [EISE92] whose operating time is less than 8ns.

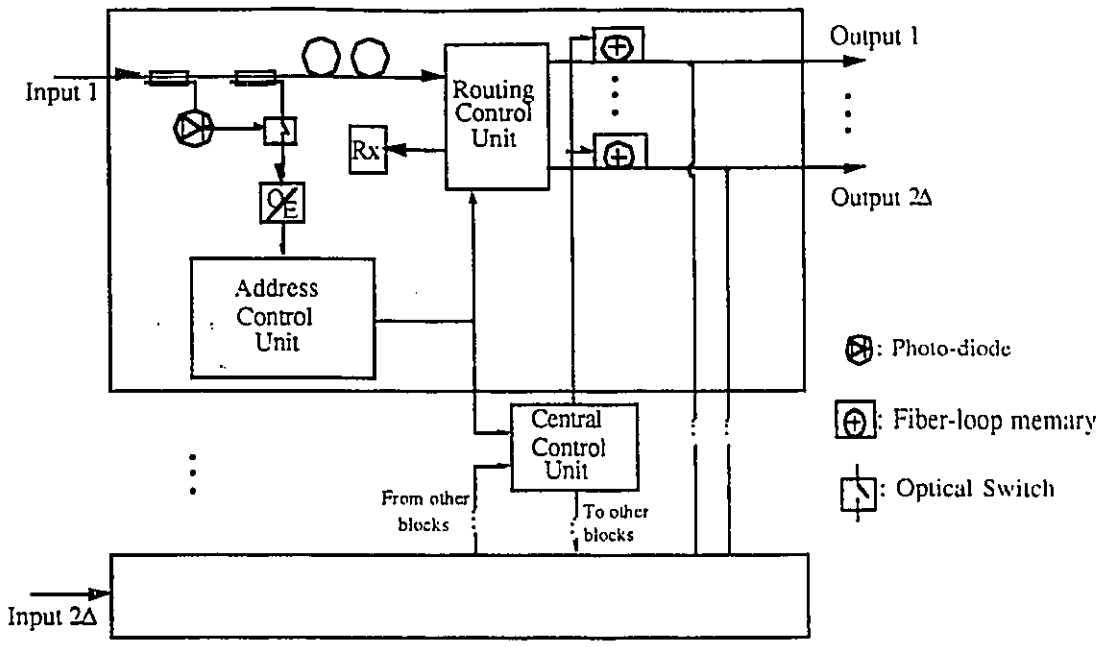


Fig. 4.4: Physical structure of a node in the (Δ,D) BdBG network. O/E: Opto-Electronic converter; Rx: Receiver.

After the opto-electronic conversion, the destination address enters the address control unit to be compared with the current node address according to the pre-chosen routing algorithm discussed in Chapter 2. The control electronic signals are sent to the routing control according to the matching results of the address. It is easy to accomplish this task in the electrical domain because VLSI circuits nowadays have been quite advanced.

After a certain delay, the packet arrives to the routing control unit which is composed of a set of optical directional couplers. Figure 4.5 shows the detail of the routing control unit. The optical directional couplers form a matrix which is controlled by the electrical signal from the address control. For example, when the destination address is totally matched with the current node address, DC1 will be set to the position such that the packet will be routed to the receiver. If the packet is for the node rather than the current node, DC2, DC3 and DC4 will be set to a proper position according to the decision of the address control.

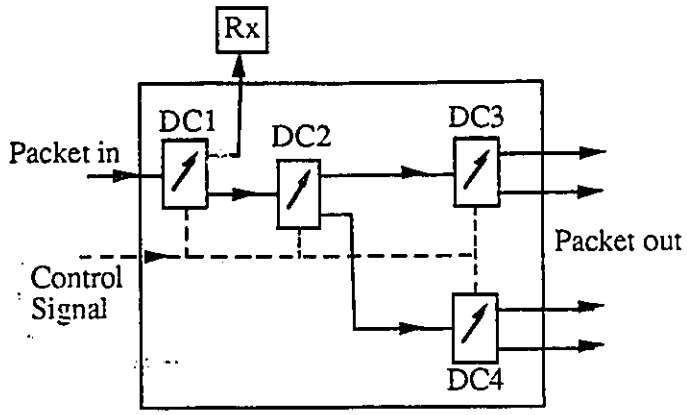


Fig. 4.5. Scheme of the Routing control. DC: 1X2 optical directional coupler.

Because there are 2Δ identical blocks for receiving and transmitting the packets, there could be collisions if more than one blocks want to transmit to the same outgoing link. To prevent this, an optical buffer made of fiber-loop memories introduced in Section 4.2.2 is used at each outgoing link and controlled by a central control unit. Each node has only one central control unit that collects information from all the functional blocks. Therefore it can decide which block has the priority to send if there is a contention at the same link by controlling the gates of the fiber-loop memories in order to keep or release the packet.

Figure 4.4 gives the operation for routing transit packets. The transmission of a new local packet, i.e., the current node is the source node, is not shown for clarity purpose. This can be done by adding the transmitter after the routing control unit followed by an optical switch which directs the packet to a desired outgoing link. However, while this transmission is being processed, if a transit packet comes in and tries to transmit at the same link, the local packet must be dropped or buffered. This is because transit packets have already consumed network resources along their path to the present node, so they must be assigned higher priority. This kind of contention problems can also be solved by the central control unit.

Note that increasing the network degree Δ would require more technological complication for the devices because more functional blocks would be needed. On the other hand, increasing D would not pose any further technological difficulties because only the operation steps in the address control unit would be increased accordingly and they are done electronically.

4.4.2 Two-level BdBG Networks

The node structure in the two-level BdBG networks is almost the same as the one-level networks except that the addressing scheme in the header of the packet is different in order to adapt to the two-level network.

As discussed in Section 4.3, there are x bytes in the header and one part of them is used to store the destination node address. However, because there are two parts of information in the address which correspond to the cluster address and the node address in the local cluster, the address control unit needs to distinguish the first part and the second part in order to make a proper routing decision. On the other hand, the routing process is faster because of fewer bits to be examined to lead to the decision. For example, if the first part of the destination address does not match the current cluster, the packet would certainly be routed towards the bridge node. It is not necessary to continue to check the second part of the address.

For the bridge nodes in hierarchical networks, their structures are similar to the ones of other nodes except that it would require more functional blocks in the node because the degree of the node is different. For example, in a $(\Delta_L,D_L)/(\Delta_U, D_U)$ BdBG network, there are $2\Delta_L+2\Delta_U$ functional blocks in the bridge node.

4.5 Design Criteria and Procedures for dBG Networks

Summarizing the performance studies of the dBG networks in Chapters 2, 3 and the physical implementation in this chapter, we provide some valuable points when designing de Bruijn Graph Networks.

4.5.1 Types of networks

Given N, which is the number of nodes expected in the network, we need to decide what type of network we should have; i.e., the network parameter Δ and D. Furthermore, we need to decide the level of the network and whether or not the bidirectional networks are preferred.

The networks we have studied so far are all complete networks in the sense that the numbers of nodes are exactly equal to Δ^D, where Δ and D are integers larger or equal to 2. Ideally, the desirable network is also the case and that will simply give us the value of Δ and D. Most often in reality, we would encounter an irregular number of nodes. Therefore, it is necessary to construct an incomplete network which has less number of nodes than the Δ^D de Bruijn Graph. This would not be difficult if we set up some pseudo-nodes in appropriate position or embedded to existing nodes. These pseudo-nodes would not receive any information rather than just simply route the packet to an outgoing link. They can consider as potential nodes in future growth as well.

Since BdBG networks have certain advantages over UdBG networks, such as throughput improvement, they should be first considered when designing a network.

The complexity of physical implementation in the node highly depends on the network degree Δ. If the cost of the optical devices plays an important role in design, a network with lower degree should be considered. Since our studies also show that the

binary networks perform well in hierarchical structures, we should choose Δ=2 in most cases. Then D can be chosen according to the number of nodes of the network.

To decide the network to be one-level or hierarchical, we need to examine the locality among nodes and also their physical locations. If significantly more traffic among close nodes than traffic among nodes far away in between, the locality factor is large. If the natural location of the nodes scatters as groups, the propagation between two groups may be much larger than that inside the group. As we discussed in Chapter 3, all these situations lead us to the decision of constructing a hierarchical network. Our results show that networks with Δ=2 to be the best choice. Not only it has the simpler hardware implementation, but also no locality is required to take advantage of the hierarchical network. For the network with 1024 nodes, (2,2)/(2,8) has the best performance in terms of network mean path length and the delay. For other types of clustering, the design has to base on locality and the propagation factor as discussed.

4.5.2 Routing Algorithms

All proposed routing algorithms are all self-routed and simple in the sense that a node can decide which neighbor it should send to by just looking at the destination address stored in the header of the packet. From the implementation point of view, since all these routing methods can be realized easily, any one of the routing methods can be chosen by network designers.

Among all routing algorithms, SCP can be advantageously used in source routing by embedding in the packet the direction (FP or BP) a packet has been assigned to. However, this would effect the header pattern which was defined early in this Chapter. Therefore, although we can use SCP routing algorithm in the networks with a small number of nodes for the sake of simplicity, the other routing methods would give us more consistency when considering future network growth. For larger networks, in order to get lower delay, RFR and NSC routing algorithms are recommended. Notice that the PMC method has worse throughput performance; therefore, it may not be used in the second

level of the hierarchical networks since the traffic intensity is much higher in the second level than in the first level.

When designing hierarchical networks, SCP can be used in the first level for small clusters. However, care must be exercised on the edge load of the second level links in order not to deteriorate the delay performance significantly. Therefore, routing methods which have higher throughput performance, such as RFR and NSC, should be used in the upper-level of the hierarchical networks.

4.5.3 Choosing Bridge Nodes

If the desired network is hierarchical, designers need to consider choosing appropriate bridge nodes in order to get the better performance.

After deciding which routing method would be used in the network, one can choose an optimum bridge node in a cluster. If a UdBG network is desired, there is a general pattern for the optimum bridge node as discussed in Chapter 3. In BdBG network case, our experience shows that nodes that locate close to the center of the logical plane layout are more likely to have smaller node mean path length; however, more exact decision can be made by computer programs. Programs for choosing the bridge node using different routing methods are included in Appendix B of this thesis.

4.5.4 Design Procedures

According to the design criteria, we propose one possible procedure when designing a MAN based on the dBG shown in Figure 4.6.

For example, given a network with 1000 nodes, one decides to use BdBG. Then Δ=2, D=10 is chosen. There are 24 pseudo nodes needed which may actually be imitated by one node. Assume locality exists among some nodes; therefore, (2,2)/(2,8) hierarchical network is chosen and the RFR routing algorithm will be applied to the network. Using the

analysis and results in Chapter 3, one should verify that the packet average delay and network throughput to check if they meet the expectation. If so, it confirms the usage of such topology. If not, one needs to reconfigure the network parameters. For example, one may change Δ to 4 and design a (4,5) BdBG network. Then after having decided all other parameters, one can check the performance and verify the decision.

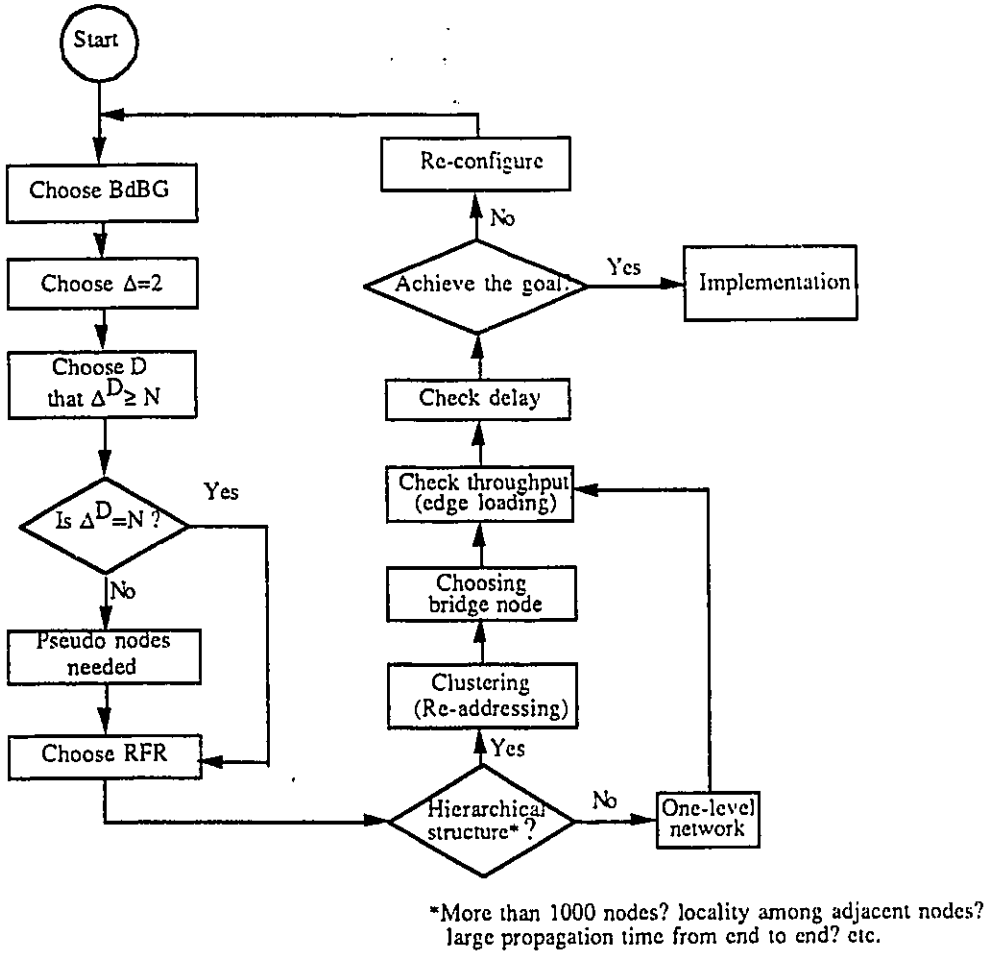


Fig. 4.6: Design procedure of dBG networks.

To interchange the normalized delay or throughput with the real packet delay and throughput, one can simply remember that the difference between these two values is a factor μC.

Figure 4.6 gives a basic idea of designing a network. The actual design procedure may be slightly different. For example, designers may decide to use hierarchical structure before choosing D. That would avoid the re-addressing step in Figure 4.6.

4.6 Discussion and Summary

In this chapter, the physical implementation method for nodes in the dBG networks is introduced. Self-routing function is made possible by converting the header of the packet into electronic signal while keeping the information payload in optical domain. Thus all nodes in the network are optically transparent. With today's technology, such implementation is feasible as shown. On the other hand, to the best of the author's knowledge, there are some difficulties to do some complicated address manipulating operations, such as pattern matching, by all optical devices with today's technologies. Consequently, purely optic networks have not been feasible at this stage. As the optical technologies are making progress every second, we believe that one day the address control can be done in the optical domain so that the network becomes purely all optical and will be more powerful.

Although the assumption of fixed packet length has been made that may conflict with the assumption we made in the analysis of previous chapters, the M/M/1 model always has the worse performance compared to the M/D/1 model. Therefore, we can assume that the actual network performance is always better than ones in the analysis. Consequently, our results are still valuable as they at least give us the basic idea of the real networks.

The general design criteria that help designers to construct a dBG network are also given. By summarizing the analysis results in previous chapters, some important points regarding the design issues have been raised.

Chapter 5
Conclusion

The third generation of optical networks requires totally new approaches in order to take the advantage of the vast optical bandwidth. One such example is the multihop system that applies the WDM technique as the access method. Good topologies for multihop systems must be capable of accommodating a large number of nodes, admit easy routing algorithms, and have low mean path length, minimum packet delay and high throughput.

This thesis proposed three new routing algorithms which can be applied to the BdBG networks. The analysis results show that all these routing algorithms have positive impact on the network performance to different extents. The RFR method was shown to be more promising because it can achieve a lower packet delay and a higher throughput while it is less complex. The other methods may still be used under certain circumstances if they are appropriate.

For designing very large networks, the hierarchical structures of the dBG networks have been proposed in the thesis. By taking into account the locality factor as well as the propagation time among nodes, hierarchical networks have certain improvement over the one-level counterparts. However, in some cases, the hierarchical BdBG networks can perform better than the one-level network even though the locality and propagation time are not considered. The results in the thesis show that by constructing the hierarchical networks, the packet average delay and throughput can be improved effectively as the locality factor increases and the propagation time among nodes is large.

The actual physical implementation of the node structure has also been studied in this thesis. With some basic optical components and devices that are available nowadays, the proposed structure can make the node optically transparent in the sense that the infor-

mation payloads of packets do not need to be converted to an electronic signal to do the routing processes. Furthermore, it is easy to adapt the node structure to all optical future networks once technology permits.

The basic design criteria and procedures have been discussed for the interested network designers.

In conclusion, the results in this thesis show that the proposed network topologies with different routing algorithms can be considered as possible candidates when constructing optical MANs. We hope that they will be given serious consideration in the process of developing the third generation optical networks.

Potential future works may include the derivation of the close form of the mean path length of the dBG networks, mathematical prove of the fact that all routing algorithms are loop-free, and various bridge node locations such as making use of the extremity nodes.

Bibliography

[ACAM87] A.S.Acampora, M.J.Karol and M.G.Hluchyj, "Terabit Lightwave Networks: The Multihop Approach," *AT&T Technical Journal*, pp.21-34, Nov.1987.

[ACAM89] A.S.Acmapora and M.J.Karol, "An overview of Lightwave Packet Networks", *IEEE Network Mag.*, pp.29-41, Jan. 1989.

[ACAM91] A.S.Acmapora and S.I.A.Shah, "Multihop Lightwave Networks: A Comparison of Store-and-Forward and Hot-Potato Routing," *Proc. IEEE INFOCOM'91*, pp.10-19, Apr.1991.

[AYAD93] F.Ayadi, J.F.Hayes and M.Kavehrad, "A WDM Cross-Connect Star Topology for the Bilayered ShuffleNet," *Proc. of 1993 Canadian Conf. on Elect. and Comp. Engineering*, Sept.1993.

[BERM89] J-C.Bermond and C.Peyrat, "de Bruijn and Kautz networks: a competitor for the hypercube?" *Hypercube and Distributed Computers*, F.Andre and J.P.Verjus (Editors), pp.279-293, 1989.

[BRAC90] C.A.Brackett, "Dense Wavelength Division Multiplexing Networks: Principles and Applications," *IEEE J. Select. Areas Commun.*, vol.8, pp.947-964, Aug.1990.

[CALZ93] M.Calzavara et al., "Optical-fiber-loop memory for multiwavelength packet buffering in ATM switching applications," *Proc. OFC/IOOC'93*, paper WD4, pp.19-20, Feb. 1993, San Jose, CA, USA.

[CAVA90] J.A.Cavailles and M.Erman, "Very Low Power Nonlinear Directional Coupling in a p-i(MQW)-n Vertical Coupler Using an Electrooptic Feedback," *IEEE Photon. Technol. Lett.*, vol.2 no.5, pp. 343-345, May 1990.

[CHLA88] I.Chlamtac and A.Ganz, "A Multibus Train Communication Architecture for High-Speed Fiber Optic Networks," *IEEE J. Select. Areas Commun.*, vol.SAC-6, pp.903-912, July 1988.

[DAND90] S.P.Dandamudi and D.L.Eager, "Hierarchical Interconnection Networks for Multicomputer Systems", *IEEE Trans. Comput.*, vol.39, no.6, pp.786-797, Jun.1990.

[DAND90] S.P.Dandamudi, "Performance analysis of a class of hierarchical hypercube multicomputer networks," *Performance Evaluation 13*, pp.159-179, 1991.

[DOWD91] P.W.Dowd, "Random Access Protocols for High Speed Inter-processor Communication Based on an Optical Passive Star Topology," *J. Lightwave Tech.*, vol.9, pp.799-808, June 1991.

[EISE92] M.Eiselt, W.Pieper and H.G.Weber, "Photonic packet switching using cascaded semiconductor optical amplifier gates," *Proc. OFC'92*, paper WH3, pp.130, Feb. 1992.

[ESFA85] H.Esfahanian and S.L.Hakimi, "Fault-Tolerant Routing in DeBruijn Communication Networks," *IEEE Trans. Comput.*, vol. C-34, no.9, pp.777-788, Sept.1985.

[GIDR91] R.Gidron and A.Temple, "TeraNet: A Multihop Multichannel ATM Lightwave Network," *Proc. ICC*, pp.602-608, 1991.

[GREE91] P.E.Green, "The Future of Fiber-Optic Computer Networks," *IEEE Comp. Mag.*, vol.24, no.9, pp.78-87, Sept.1991.

[GREE93] P.E.Green, *Fiber Optic Networks*, Prentice Hall, New Jersey, 1993.

[IKED83] M.Ikeda, "Switching characteristics of laser diode switch," *IEEE J. Quantum Electron.*, vol.QE-19, no.2, pp.157-164, 1983.

[IMAS85] M.Imase, T.Soneoka and K.Okada, "Connectivity of Regular Directed Graphs with Small Diameters," *IEEE Trans. Comput.*, vol.C-34, no.3, pp.267-273, Mar.1985.

[KARO91] M.J.Karol and S.Z.Shaikh, "A Simple Adaptive Routing Scheme for Congestion Control in ShuffleNet Multihop Lightwave Networks." *IEEE J. Select. Areas Commun.*, vol.9, pp.1040-1051, Sept.1991.

[KESS91] G.C.Kessler and D.A.Train, *Metropolitan Area Networks*, McGraw-Hill, Inc., 1991.

[KLEI64] L.Kleinrock, *Communication Nets; Stochastic Message Flow and Delay*, McGraw-Hill (New York), 1964. Reprinted by Dover Publications, 1972.

[KLE'75] L.Kleinrock, *Queueing Systems*, vol.1 & 2, John Wiley & Sons, Inc., 1975.

[KRIS90] A.Krishna and B.Hajek, "Performance of Shuffle-like Switching Networks with Deflection," *Proc. IEEE INFOCOM'90*, pp.473-480, June 1990.

[LEMP70] A.Lempel, "On a Homomorphism of the de Bruijn Graph and Its Applications to the Design of Feedback Shift Registers," *IEEE Trans. Comput.*, vol.C-19, no.12, pp.1204-1209, Dec.1970.

[LIB92] B.Li and A.Ganz. "Virtual Topologies for WDM Star LANs—The Regular Structure Approach," *Proc. IEEE INFOCOM'92*, pp.2134-2143, May 1992.

[MAXE85] N.F.Maxemchuk, "Regular Mesh Topologies in Local and Metropolitan Area Networks," *AT&T Technical Journal*, vol.64, no.7, Sept.1985, pp.1659-1685.

[MAXE87] N.F.Maxemchuk, "Routing in the Manhattan Street Network," *IEEE Trans. Commun.*, vol.COM-35, pp.503-512, May 1987.

[MUK92a] B.Mukherjee, "WDM-Based Local Lightwave Networks, Part I : Single hop systems," *IEEE Network Mag.*, May. 1992, pp.12-27.

[MUK92b] B.Mukherjee, "WDM-Based Local Lightwave Networks, Part II: Multihop systems," *IEEE Network Mag.*, pp.20-32, July 1992.

[OKAY91] H.Okayama, T.Ushikubo and T.Ishida, "Directional Coupler Switch with Reduced Voltage-length Product," *J. Lightwave Tech.*, vol.9, no.11, pp. 1561-1566, Nov.1991.

[PRAD82] D.K.Pradhan and S.M.Reddy, "A Fault-Tolerant Communication Architecture for Distributed Systems," *IEEE Trans. Comput.*, vol.C-31, no.9, pp.863-870, Sept.1982.

[ROWL93] R.A.Rowley and B.Bose, "Fault-Tolerant Ring Embedding in de Bruijn Networks," *IEEE Trans. Comput.*, vol.42, no.12, pp.1480-1486, Dec.1993.

[SAMA89] M.R.Samatham and D.K.Pradhan, "The De Bruijn Multiprocessor Network: A versatile Parallel Processing and Sorting Network for VLSI," *IEEE Trans. Comput.*, vol.38, no.4, pp.567-581, Apr.1989.

[SIVA91] K.Sivarajan and R.Ramaswami, "Multihop Lightwave Networks Based On De Bruijn Graphs," *Proc. IEEE INFOCOM'91*, pp.1001-1011, Apr.1991.

[SIVA92] K.Sivarajan, "Multihop Logical Topologies For Gigabit Lightwave Networks," *IEEE LTS Mag.*, pp.20-25, Aug.1992.

[SIVA94] K.Sivarajan and R.Ramaswami, "Lightwave Networks Based on de Bruijn Graphs," *IEEE/ACM Trans. on Networking*, vol.2, no.1, Feb.1994.

[SRID87] M.A.Sridhar and C.S. Raghavendra, "Uniform minimal full-access networks," *Proc. Internat. Conf. on Parallel Processing*, pp.401-406, 1987.

[SRID88] M.A.Sridhar, "On the Connectivity of the De Bruijn Graph," *Information Processing Letters* , vol.27, no.6, pp.315-318, 13 May 1988.

[SRID91] M.A.Sridhar and C.S. Raghavendra, "Fault-Tolerant Networks Based on the de Bruijn Graph," *IEEE Trans. Comput.*, vol.40, no.10, pp.1167-1174, Oct.1991.

[SRID92] M.A. Sridar, "The Undirected de Bruijn Graph: Fault Tolerance and Routing Algorithms," *IEEE Trans. on Circuits and Systems-1:Fundamental Theory and Applications"*, vol. 39, no. 1, pp.45-48, Jan. 1992.

[SUDH91] G.N.M.Sudhakar, M.Kavehrad and N.Georganas, "Multi-Control Channel Very High Speed Optical Fiber Local Area Networks and Their Interconnections Using Passive Star Topology," *Proc. IEEE GLOBECOM'91*, pp.624-628, Dec.1991.

[ZHAN90] Z.Zhang and A.S.Acampora, "Analysis of Multihop Lightwave Networks," *Proc. IEEE GLOBECOM'90*, Dec.1990.

[ZHAN91] Z.Zhang and A.S.Acampora, "Performance Analysis of Multihop Lightwave Networks with Hot-potato Routing and Distance-Age-Priorities," *Proc. IEEE INFOCOM'91*, pp.1012-1021, Apr.1991.

✓ PAGINATION ERROR.

TEXT COMPLETE.

✓ERREUR DE PAGINATION.

LE TEXTE EST COMPLET.

NATIONAL LIBRARY OF CANADA.

CANADIAN THESES SERVICE.

BIBLIOTHEQUE NATIONALE DU CANADA.

SERVICE DES THESES CANADIENNES.

Appendix

Analysis programs

Generally, to run these programs given in the Appendix, one should input some proper parameters such as network degree Δ and locality factor alpha if needed. The normalized offered loads can also be put into the command line arguments to run the programs. The network diameter D, the desired bridge node address are required to be changed inside the programs. Furthermore, the bunch of "for loops" at the beginning of each program needs to be changed according to the DIM. For example, if DIM=5, the for loops should have 10 lines in which the s[] and d[] should range from 0 to 4. Details of programs are listed below. For the sake of simplicity, the comments are only given in the first program "MPL_SCP". In other programs, only the differences are explained.

A. Programs for One-level BdBG Networks

```
/*The following program calculates the Mean Path Length of the Bidirectional de Bruijn Graph using the "SCP"
routing algorithm.

                Program name: MPL_SCP.                                  Oct.28, 1993.
                By Zhi Feng.
*/
```

```
#define DIM 6                       /* Network diameter */
#define NODE 1024                   /* Number of nodes  */

#include "stdlib.h"
#include "math.h"
#include "string.h"

int dlt,                            /* Network degree Δ */
    *left(int *n1, int *n2),    /* Address of the left neighbor */
    *right(int *n1, int *n2),       /* Address of the right neighbor */
   next[DIM],                       /* Address of next node */
   mergefwd(int *s, int *d),        /* Required hops using FP */
   mergebwd(int *s, int *d),        /* Required hops using BP */
  load[NODE][NODE];         /* Edge load between two nodes */
void check(int *ss, int *dd),       /* Send one packet from sourc to destination */
     record(int *n1, int *n2);      /* Record the edge load */
void send_fwd(int *sc, int *de),    /* Using FP */
     send_bwd(int *sc, int *de);    /* Using BP */
void strxfrm_mod(char *str1, char *str2, int count);      /*See in function below*/

char atoc[]="0123456789";
```

```
void main(argc,argv)
int argc;
char *argv[];
{
register int s[DIM], d[DIM];
int i, j;
long max=0, sum=0, node;
double mpath;

   dlt=atoi(argv[1]);
   printf("delta=%d dimension=%d\n",dlt,DIM);
   node = pow((double)dlt,(double)DIM);

/* The following "for" loops have to cooperated with the number "DIM". For example, the following one is for
DIM = 6. */
    for (s[0]=0;s[0]<=dlt-1;s[0]++)
    for (s[1]=0;s[1]<=dlt-1;s[1]++)
    for (s[2]=0;s[2]<=dlt-1;s[2]++)
    for (s[3]=0;s[3]<=dlt-1;s[3]++)
    for (s[4]=0;s[4]<=dlt-1;s[4]++)
    for (s[5]=0;s[5]<=dlt-1;s[5]++)
  /*  for (s[6]=0;s[6]<=dlt-1;s[6]++)
    for (s[7]=0;s[7]<=dlt-1;s[7]++)
    for (s[8]=0;s[8]<=dlt-1;s[8]++)
    for (s[9]=0;s[9]<=dlt-1;s[9]++)
    for (d[0]=0;d[0]<=dlt-1;d[0]++)
    for (d[1]=0;d[1]<=dlt-1;d[1]++)
    for (d[2]=0;d[2]<=dlt-1;d[2]++)
    for (d[3]=0;d[3]<=dlt-1;d[3]++)
    for (d[4]=0;d[4]<=dlt-1;d[4]++)
    for (d[5]=0;d[5]<=dlt-1;d[5]++)
  /*  for (d[6]=0;d[6]<=dlt-1;d[6]++)
    for (d[7]=0;d[7]<=dlt-1;d[7]++)
    for (d[8]=0;d[8]<=dlt-1;d[8]++)
    for (d[9]=0;d[9]<=dlt-1;d[9]++)  */              check(s,d);

    for (i=0; i<=node-1; i++)
    for (j=0; j<=node-1; j++)  {
      if  (max<load[i][j])
        max = load[i][j];
        sum += load[i][j];
    }

     mpath=(float)sum/(node*(node-1.));
    printf("maximun edge loading=%d  sum=%d\n",max, sum);
     printf("mpath=%f\n",mpath);
}

/* The "check" imitates the operation of sending a packet from source node denoted by (ss) to the destination node
denoted by (dd). */
void check (int *ss, int *dd)
{
int i, pathfwd, pathbwd;
char str_ss[DIM+1], str_dd[DIM+1];

memset(str_ss,'\0',sizeof(str_ss));
memset(str_dd,'\0',sizeof(str_dd));
```

```
  for (i=0;i<=DIM-1;i++) {
      str_ss[i]=atoc[*(ss+i)];
     str_dd[i]=atoc[*(dd+i)];
  }

  if (strcmp(str_ss, str_dd) == 0) ;
  else {

/* According to SCP, required hops using FP and BP are compared and then the shorter one is chosen. */
     pathfwd = mergefwd(ss, dd);
     pathbwd = mergebwd(ss, dd);

     if (pathfwd > pathbwd) send_fwd(ss, dd);
     else if (pathfwd < pathbwd) send_bwd(ss, dd);
     else
       if ((random()&01) == 0) send_fwd(ss, dd);
       else send_bwd(ss, dd);
  }
}

/* The "mergefwd" returns the matched number of two addresses using FP. */
int mergefwd(int *s, int *d)
{
int i;
char str_s[DIM+1], str_d[DIM+1],
     str_tmp1[DIM+1], str_tmp2[DIM+1];

  for (i=0;i<=DIM-1;i++) {
      str_s[i]=atoc[*(s+i)];
      str_d[i]=atoc[*(d+i)];
  }

  for (i=0;i<=DIM-1;i++) {
     strxfrm(str_tmp1, str_s+i, DIM-i);
     strxfrm_mod(str_tmp2, str_d, DIM-i);

     if (strcmp(str_tmp1, str_tmp2) == 0)
       return (strlen(str_tmp1));
  }
  return 0;
}

/* The "mergebwd" returns the matched number of two addresses using BP. */
int mergebwd(int *s, int *d)
{
int i;
char str_s[DIM+1], str_d[DIM+1],
     str_tmp1[DIM+1], str_tmp2[DIM+1];

  for (i=0;i<=DIM-1;i++) {
      str_s[i]=atoc[*(s+i)];
      str_d[i]=atoc[*(d+i)];
  }

  for (i=0;i<=DIM-1;i++) {
     strxfrm_mod(str_tmp1, str_s, DIM-i);
     strxfrm(str_tmp2, str_d+i, DIM-i);

     if (strcmp(str_tmp1, str_tmp2) == 0)
```

```
        return (strlen(str_tmp1));
    }
    return 0;
}

/* The "send_fwd" checks the desired left neighbor and records the edge load then repeats until the packet reaches
the destination.    */
void send_fwd(int *sc, int *de)
{
int i, *Ptrnext, Tmpst[DIM];
char str_sc[DIM+1], str_de[DIM+1];

  for (i=0;i<=DIM-1;i++) {
      str_sc[i] = atoc[*(sc+i)];
      str_de[i] = atoc[*(de+i)];

      Tmpst[i] = *(sc+i);
  }

  if (strcmp(str_sc, str_de) == 0) ;
  else {
     Ptrnext = left(Tmpst, de);
     record(Tmpst, Ptrnext);
     send_fwd(Ptrnext, de);
  }
}

/* The "send_bwd" checks the desired right neighbor and records the edge load then repeats until the packet
reaches the destination.    */
void send_bwd(int *sc, int *de)
{
int i, *Ptrnext, Tmpst[DIM];
char str_sc[DIM+1], str_de[DIM+1];

   for (i=0;i<=DIM-1;i++) {
       str_sc[i] = atoc[*(sc+i)];
       str_de[i] = atoc[*(de+i)];

       Tmpst[i] = *(sc+i);
   }

   if (strcmp(str_sc, str_de) == 0) ;
   else {
      Ptrnext = right(Tmpst, de);
      record(Tmpst, Ptrnext);
      send_bwd(Ptrnext, de);
   }
}

/* This function returns the address of the left neighbor according to the current and destination address. */
int *left(int *n1, int *n2)
{
int i, path;

   path = mergefwd(n1, n2);

   for (i=0; i<=DIM-2; i++)
      next[i] = *(n1+i+1);
```

```
  next[DIM-1] = *(n2+path);

  return (next);
}

/* This function returns the address of the right neighbor according to the current and destination address. */
int *right(int *n1, int *n2)
{
int i, path;

  path = mergehwd(n1, n2);

  next[0] = *(n2+DIM-path-1);

  for (i=1; i<=DIM-1; i++)
    next[i] = *(n1+i-1);

  return (next);
}

/* Function "strxfrm_mod" transforms the first "count" characters of the string pointed to by str2 then puts the
result into the string pointed to by str1 which ended with a null pointer. */
void strxfrm_mod(char *str1, char *str2, int count)
{
int i;
  for (i=0;i<count;i++)
    *(str1+i)= *(str2+i);

  *(str1+count)='\0';
}


/* Function "record" records the traffic loads between the nodes pointed to by n1 and n2.    */
void record(int *n1, int *n2)
{
  ++load[conv(n1)][conv(n2)];
}

/* Function "conv" converts the digit pointed to by n into a decimal digit and then return it.        */
int conv(int *n)
{
register int i;
long sum;

  for (sum=0, i=0; i<=DIM-1; i++)
    sum += (*(n+DIM-i-1))*(long)pow((double)dlt,(double)i);
  return (sum);
}
```

```
/* The following program calculates the Mean Path Length of the Bidirectional de Bruijn Graph using the "RFR"
routing algorithm.

                Program name: MPL_RFR.                                          Nov.16, 1993.
                By Zhi Feng.
*/

#define DIM 6                   /* Network diameter */
#define NODE 1024               /* Number of nodes  */

#include "stdlib.h"
#include "math.h"
#include "string.h"

int dlt, next[DIM];
int *left(int *n1, int *n2), *right(int *n1, int *n2);
int mergefwd(int *s, int *d), mergebwd(int *s, int *d);
int load[NODE][NODE];
void strxfrm_mod(char *str1, char *str2, int count);
void check(int *ss, int *dd), record(int *n1, int *n2);
void send_fwd(int *sc, int *de), send_bwd(int *sc, int *de);

char atoc[]="0123456789";

void main(argc,argv)
int argc;
char *argv[];
{
register int s[DIM], d[DIM];
int i, j;
long max=0, sum=0, node;
double mpath;

  dlt=atoi(argv[1]);
  printf("delta=%d dimension=%d\n",dlt,DIM);
  node = pow((double)dlt,(double)DIM);

  for (s[0]=0;s[0]<=dlt-1;s[0]++)
  for (s[1]=0;s[1]<=dlt-1;s[1]++)
  for (s[2]=0;s[2]<=dlt-1;s[2]++)
  for (s[3]=0;s[3]<=dlt-1;s[3]++)
  for (s[4]=0;s[4]<=dlt-1;s[4]++)
  for (s[5]=0;s[5]<=dlt-1;s[5]++)
/*  for (s[6]=0;s[6]<=dlt-1;s[6]++)
  for (s[7]=0;s[7]<=dlt-1;s[7]++)
  for (s[8]=0;s[8]<=dlt-1;s[8]++)
  for (s[9]=0;s[9]<=dlt-1;s[9]++)  */
  for (d[0]=0;d[0]<=dlt-1;d[0]++)
  for (d[1]=0;d[1]<=dlt-1;d[1]++)
  for (d[2]=0;d[2]<=dlt-1;d[2]++)
  for (d[3]=0;d[3]<=dlt-1;d[3]++)
  for (d[4]=0;d[4]<=dlt-1;d[4]++)
  for (d[5]=0;d[5]<=dlt-1;d[5]++)
/*  for (d[6]=0;d[6]<=dlt-1;d[6]++)
  for (d[7]=0;d[7]<=dlt-1;d[7]++)
  for (d[8]=0;d[8]<=dlt-1;d[8]++)
  for (d[9]=0;d[9]<=dlt-1;d[9]++)  */

    check(s,d);
```

```
  for (i=0; i<=node-1; i++)
  for (j=0; j<=node-1; j++) {
    if (max<load[i][j])
      max = load[i][j];
      sum += load[i][j];
  }

   mpath=(float)sum/(node*(node-1.));
  printf("maximun edge loading=%d  sum=%d\n",max, sum);
   printf("mpath=%f\n",mpath);
}

void check (int *ss, int *dd)
{
int i, pathfwd, pathbwd;
char str_ss[DIM+1], str_dd[DIM+1];

    memset(str_ss,'\0',sizeof(str_ss));
   memset(str_dd,'\0',sizeof(str_dd));

  for (i=0;i<=DIM-1;i++) {
      str_ss[i]=atoc[*(ss+i)];
     str_dd[i]=atoc[*(dd+i)];
  }
  if (strcmp(str_ss, str_dd) ==  0) ;
  else {

    pathfwd = mergefwd(ss, dd);
    pathbwd = mergebwd(ss, dd);

    if (pathfwd > pathbwd) send_fwd(ss, dd);
    else if (pathfwd < pathbwd) send_bwd(ss, dd);
    else
      if ((random()&01) == 1) send_fwd(ss, dd);
      else send_bwd(ss, dd);
  }
}

int mergefwd(int *s, int *d)
{
int i;
char str_s[DIM+1], str_d[DIM+1],
     str_tmp1[DIM+1], str_tmp2[DIM+1];

    memset(str_s,'\0',sizeof(str_s));
   memset(str_d,'\0',sizeof(str_d));

  for (i=0;i<=DIM-1;i++) {
      str_s[i]=atoc[*(s+i)];
     str_d[i]=atoc[*(d+i)];
  }

  for (i=0;i<=DIM-1;i++) {
     strxfrm(str_tmp1, str_s+i, DIM-i);
     strxfrm_mod(str_tmp2, str_d, DIM-i);
     if (strcmp(str_tmp1, str_tmp2) == 0)
       return (strlen(str_tmp1));
  }
```

```
  return 0;
}

int mergebwd(int *s, int *d)
{
int i;
char str_s[DIM+1], str_d[DIM+1], str_tmp1[DIM+1], str_tmp2[DIM+1];

    memset(str_s,'\0',sizeof(str_s));
    memset(str_d,'\0',sizeof(str_d));

  for (i=0;i<=DIM-1;i++) {
      str_s[i]=atoc[*(s+i)];
      str_d[i]=atoc[*(d+i)];
  }

  for (i=0;i<=DIM-1;i++) {
     strxfrm_mod(str_tmp1, str_s, DIM-i);
     strxfrm(str_tmp2, str_d+i, DIM-i);
     if (strcmp(str_tmp1, str_tmp2) == 0)
       return (strlen(str_tmp1));
  }
  return 0;
}

/* The "send_fwd" checks the desired left neighbor and records the edge load then returns to the check function to
let it decide the next step.  */
void send_fwd(int *sc, int *de)
{
int i, *Ptrnext, Tmpst[DIM];
char str_sc[DIM+1], str_de[DIM+1];

    memset(str_sc,'\0',sizeof(str_sc));
    memset(str_de,'\0',sizeof(str_de));

  for (i=0;i<=DIM-1;i++) {
      str_sc[i] = atoc[*(sc+i)];
      str_de[i] = atoc[*(de+i)];
      Tmpst[i] = *(sc+i);
  }

   if (strcmp(str_sc, str_de) == 0) printf("error1");
  else {
     Ptrnext = left(Tmpst, de);
     record(Tmpst, Ptrnext);
     check(Ptrnext, de);
  }
}

/* The "send_bwd" checks the desired right neighbor and records the edge load then returns to the check function
to let it decide the next step.  */
void send_bwd(int *sc, int *de)
{
int i, *Ptrnext, Tmpst[DIM];
char str_sc[DIM+1], str_de[DIM+1];

    memset(str_sc,'\0',sizeof(str_sc));
    memset(str_de,'\0',sizeof(str_de));
```

```
  for (i=0;i<=DIM-1;i++) {
      str_sc[i] = atoc[*(sc+i)];
      str_de[i] = atoc[*(de+i)];
      Tmpst[i] = *(sc+i);
  }

  if (strcmp(str_sc, str_de) == 0) printf("error2");
  else {
     Ptrnext = right(Tmpst, de);
     record(Tmpst, Ptrnext);
     check(Ptrnext, de);
  }
}

int *left(int *n1, int *n2)
{
int i, path;

  path = mergefwd(n1, n2);
  for (i=0; i<=DIM-2; i++)
     next[i] = *(n1+i+1);
  next[DIM-1] = *(n2+path);
  return (next);
}

int *right(int *n1, int *n2)
{
int i, path;

  path = mergebwd(n1, n2);
  next[0] = *(n2+DIM-path-1);
  for (i=1; i<=DIM-1; i++)
     next[i] = *(n1+i-1);
  return (next);
}

void strxfrm_mod(char *str1, char *str2, int count)
{
int i;
   for (i=0;i<count;i++)
      *(str1+i)= *(str2+i);
      *(str1+count)='\0';
}

void record(int *n1, int *n2)
{
   ++load[conv(n1)][conv(n2)];
}

int conv(int *n)
{
register int i;
long sum;

  for (sum=0, i=0; i<=DIM-1; i++)
     sum += (*(n+DIM-i-1))*(long)pow((double)dlt,(double)i);
  return (sum);
}
```

```
/* The following program calculates the Mean Path Length of the Bidirectional de Bruijn Graph using the "NSC"
routing algorithm.

                Program name: MPL_NSC.                                          Nov.16, 1993.
                By Zhi Feng.
*/

#define DIM 3
#define NODE 1024

#include "stdlib.h"
#include "string.h"
#include "math.h"

int dlt;
void check(int *s, int *d), record(int *n1, int *n2);
int *leftneigh(int *s3,int i),*rightneigh(int *s4, int i);
int conv(int *n), checknode(int *n1, int *n2);
int next[DIM],neighbor[DIM];
int load[NODE][NODE];
void strxfrm_mod(char *str1, char *str2, int count);
int mergefwd(int *s, int *d), mergebwd(int *s, int *d);

char atoc[]="0123456789";

void main(argc,argv)
int argc;
char *argv[];
{
int i, j;
register int s[DIM], d[DIM];
long max=0, sum=0, node;
double mpath;

   dlt=atoi(argv[1]);
   printf("delta=%d dimension=%d\n",dlt,DIM);
   node = pow((double)dlt,(double)DIM);

   for (s[0]=0;s[0]<=dlt-1;s[0]++)
   for (s[1]=0;s[1]<=dlt-1;s[1]++)
   for (s[2]=0;s[2]<=dlt-1;s[2]++)
  /*  for (s[3]=0;s[3]<=dlt-1;s[3]++)
   for (s[4]=0;s[4]<=dlt-1;s[4]++)
   for (s[5]=0;s[5]<=dlt-1;s[5]++)
   for (s[6]=0;s[6]<=dlt-1;s[6]++)
   for (s[7]=0;s[7]<=dlt-1;s[7]++)
   for (s[8]=0;s[8]<=dlt-1;s[8]++)
   for (s[9]=0;s[9]<=dlt-1;s[9]++)  */
   for (d[0]=0;d[0]<=dlt-1;d[0]++)
   for (d[1]=0;d[1]<=dlt-1;d[1]++)
   for (d[2]=0;d[2]<=dlt-1;d[2]++)
  /*  for (d[3]=0;d[3]<=dlt-1;d[3]++)
   for (d[4]=0;d[4]<=dlt-1;d[4]++)
   for (d[5]=0;d[5]<=dlt-1;d[5]++)
   for (d[6]=0;d[6]<=dlt-1;d[6]++)
   for (d[7]=0;d[7]<=dlt-1;d[7]++)
   for (d[8]=0;d[8]<=dlt-1;d[8]++)
   for (d[9]=0;d[9]<=dlt-1;d[9]++)  */
```

```
    check(s,d);

  for (i=0; i<=node-1; i++)
  for (j=0; j<=node-1; j++) {
    if (max<load[i][j])
      max = load[i][j];
      sum += load[i][j];
  }

   mpath=(float)sum/(node*(node-1.));
   printf("maximun edge loading=%d\n",max);
   printf("mpath=%f\n",mpath);
}

/* Function "check" gets the source and destination address codes pointed to by ss and dd, decides the next node to
be sent, records the traffic loading between the source and the next node, then recalls itself recursively by passing
the addresses of next and the destination.                    */
void check(int *ss,int *dd)
{
register int i, j;
int TmpSc[DIM];
int hops=DIM, hops_new;
int *left, *right;
char str_s[DIM+1], str_d[DIM+1];

  for (i=0;i<=DIM-1;i++) TmpSc[i]= *(ss+i);

   memset(str_s,'\0',sizeof(str_s));
   memset(str_d,'\0',sizeof(str_d));

  for (i=0;i<=DIM-1;i++) {
      str_s[i]=atoc[*(TmpSc+i)];
      str_d[i]=atoc[*(dd+i)];
  }

   if (strcmp(str_s,str_d)==0) ;
   else {

/* Searching for a left neighbor that has minimum hops to the destination.   */
     for (i=0;i<=dlt-1;i++) {
        left=leftneigh(TmpSc,i);
        hops_new=checknode(left,dd);

        if (hops_new > hops) ;
        else if (hops > hops_new) {
          hops = hops_new;
           for (j=0;j<=DIM-1;j++) next[j]= *(left+j);
       }
       else
                if ((random()&01)==0)
            for (j=0;j<=DIM-1;j++) next[j]= *(left+j);
    }

/* Searching for a right neighbor that has minimum hops to the destination.   */
     for (i=0;i<=dlt-1;i++) {
        right=rightneigh(TmpSc,i);
        hops_new=checknode(right,dd);

        if (hops_new > hops) ;
```

```
      else if (hops > hops_new) {
        hops = hops_new;
        for (j=0;j<=DIM-1;j++) next[j]= *(right+j);
      }
      else
          if ((random()&01)==0)
           for (j=0;j<=DIM-1;j++) next[j]= *(right+j);
    }

/* Call the function "record" to record the traffic between source and the next node.                    */
      record(TmpSc,next);

/* Substitude the source by "next", recall the function "check"
   until the destination is reached.                          */
      check(next,dd);
    }
  }

/* The checknode function checks the required FP or BP from the node pointed by n1 to node pointed by n2.    */
int checknode(int *n1, int *n2)
{
int pathfwd, pathbwd, temp;

     pathfwd = mergefwd(n1, n2);
     pathbwd = mergebwd(n1, n2);

     if (pathfwd <= pathbwd) temp = DIM-pathbwd;
     else if (pathfwd > pathbwd) temp = DIM-pathfwd;

   return(temp);
 }

 int mergefwd(int *s, int *d)
 {
 int i;
 char str_s[DIM+1], str_d[DIM+1],
      str_tmp1[DIM+1], str_tmp2[DIM+1];

    for (i=0;i<=DIM-1;i++) {
        str_s[i]=atoc[*(s+i)];
        str_d[i]=atoc[*(d+i)];
    }

    for (i=0;i<=DIM-1;i++) {
       strxfrm(str_tmp1, str_s+i, DIM-i);
       strxfrm_mod(str_tmp2, str_d, DIM-i);

       if (strcmp(str_tmp1, str_tmp2) == 0)
         return (strlen(str_tmp1));
    }
    return 0;
  }

  int mergebwd(int *s, int *d)
  {
  int i;
  char str_s[DIM+1], str_d[DIM+1],
       str_tmp1[DIM+1], str_tmp2[DIM+1];
```

```
 for (i=0;i<=DIM-1;i++) {
     str_s[i]=atoc[*(s+i)];
     str_d[i]=atoc[*(d+i)];
 }
 for (i=0;i<=DIM-1;i++) {
    strxfrm_mod(str_tmp1, str_s, DIM-i);
    strxfrm(str_tmp2, str_d+i, DIM-i);
    if (strcmp(str_tmp1, str_tmp2) == 0)
      return (strlen(str_tmp1));
 }
 return 0;
}

/* Function "rightneigh" returns a pointer pointing to one of the right neighbors of the node pointed by s3. */
int *rightneigh(int *s3, int i)
{
int j;

 for (j=0;j<=DIM-1;j++) {
   if (j==0) neighbor[j]=i;
   else neighbor[j]= *(s3-1+j);
 }
  return (neighbor);
}

/* Function "leftneigh" returns a pointer pointing to one of the left neighbors of the node pointed by s3.    */
int *leftneigh(int *s4, int i)
{
int j;

 for (j=0;j<=DIM-1;j++) {
   if (j==(DIM-1)) neighbor[j]=i;
   else neighbor[j]= *(s4+j+1);
 }
  return (neighbor);
}

void strxfrm_mod(char *str1, char *str2, int count)
{
int i;
  for (i=0;i<count;i++)
    *(str1+i)= *(str2+i);
  *(str1+count)='\0';
}

void record(int *n1, int *n2)
{
  ++load[conv(n1)][conv(n2)];
}

int conv(int *n)
{
register int i;
long sum;

  for (sum=0, i=0; i<=DIM-1; i++)
    sum += (*(n+DIM-i-1))*(long)pow((double)dlt,(double)i);
  return (sum);
}
```

```
/* The following program calculates the Mean Path Length of the Bidirectional de Bruijn Graph using the "PMC"
routing algorithm.

                Program name: MPL_PMC.                                          Oct.27, 1993.
                By Zhi Feng.
*/
#define DIM 10
#define NODE 1024

#include "stdlib.h"
#include "string.h"
#include "math.h"

int dlt;
void check(int *s, int *d), record(int *n1, int *n2);
int *leftneigh(int *s3,int i),*rightneigh(int *s4, int i), conv(int *n), pattern(int *s2,int *d2,int *pds);
int next[DIM],neighbor[DIM], load[NODE][NODE];
char *strstr_rv(char *str1, char *str2);
void strxfrm_mod(char *str1, char *str2, int count);

char atoc[]="0123456789";

void main(argc,argv)
int argc;
char *argv[];
{
int i, j;
register int s[DIM], d[DIM];
long max=0, sum=0, node;
double mpath;

   dlt=atoi(argv[1]);
   printf("delta=%d dimension=%d\n",dlt,DIM);
   node = pow((double)dlt,(double)DIM);

   for (s[0]=0;s[0]<=dlt-1;s[0]++)        /* The sizes of s and d */
   for (s[1]=0;s[1]<=dlt-1;s[1]++)        /* should be ajusted    */
   for (s[2]=0;s[2]<=dlt-1;s[2]++)        /* according to DIM.    */
   for (s[3]=0;s[3]<=dlt-1;s[3]++)        /* This needs to be     */
   for (s[4]=0;s[4]<=dlt-1;s[4]++)        /* done by the person   */
   for (d[0]=0;d[0]<=dlt-1;d[0]++)        /* who executes this    */
   for (d[1]=0;d[1]<=dlt-1;d[1]++)        /* this program.        */
   for (d[2]=0;d[2]<=dlt-1;d[2]++)
   for (d[3]=0;d[3]<=dlt-1;d[3]++)
   for (d[4]=0;d[4]<=dlt-1;d[4]++)

     check(s,d);

   for (i=0; i<=node-1; i++)
   for (j=0; j<=node-1; j++) {
     if (max<load[i][j])
       max = load[i][j];
       sum += load[i][j];
   }

    mpath=(float)sum/(node*(node-1.));
    printf("maximun edge loading=%d\n",max);
    printf("mpath=%f\n",mpath);
}
```

```
void check(int *ss,int *dd)
{
register int i, j;
int TmpSc[DIM];
int match, match_new, dist=0, dist_new=0, dstnc;
int *left, *right;
char str_s[DIM+1], str_d[DIM+1];

  for (i=0;i<=DIM-1;i++) TmpSc[i]= *(ss+i);
  for (i=0;i<=DIM-1;i++) {
      str_s[i]=atoc[*(TmpSc+i)];
      str_d[i]=atoc[*(dd+i)];
  }
   if (strcmp(str_s,str_d)==0) ;
  else {

/* Call the function "pattern" which returns the number of digits that match the two string pointed to by TmsSc
and dd, also returns the distance pointed to by &dist.                    */
      match=pattern(TmpSc,dd,&dist);
     dstnc = dist;

/* Choose the next node which has largest matched pattern or same matched pattern but larger distance.          */
     for (i=0;i<=dlt-1;i++) {
        left=leftneigh(TmpSc,i);
        match_new=pattern(left,dd,&dist_new);
       if (match_new<match) ;
       else if (match_new>match) {
         match=match_new;
         dstnc=dist_new;
          for (j=0;j<=DIM-1;j++) next[j]= *(left+j);
       }
       else {
         if (dist_new<dstnc) ;
         else if (dist_new>dstnc) {
           dstnc=dist_new;
            for (j=0;j<=DIM-1;j++) next[j]= *(left+j);
         }
         else
                if ((random()&01)==0)
            for (j=0;j<=DIM-1;j++) next[j]= *(left+j);
       }
    }

     for (i=0;i<=dlt-1;i++) {
        right=rightneigh(TmpSc,i);
        match_new=pattern(right,dd,&dist_new);

       if (match_new<match) ;
       else if (match_new>match) {
         match=match_new;
         dstnc=dist_new;
          for (j=0;j<=DIM-1;j++) next[j]= *(right+j);
       }
       else {
         if (dist_new<dstnc) ;
         else if (dist_new>dstnc) {
           dstnc=dist_new;
            for (j=0;j<=DIM-1;j++) next[j]= *(right+j);
```

```
      }
      else
        if ((random()&01)==0)
         for (j=0;j<=DIM-1;j++) next[j]= *(right+j);
    }
  }

/* Call the function "record" to record the traffic between source and the next node.          */
    record(TmpSc,next);

/* Substitude the source by next, recall the function "check" until the destination is reached.  */
  check(next,dd);
 }
}

/* Function "pattern" compares two strings pointed to by s2 and d2, and returns the number of the largest matched
pattern and also the distance between the positions in two strings.          */
int pattern (int *s2, int *d2, int *pds)
{
register int i, j;
int match, dist, len1=0, len2=0;
char str1[DIM+1], str2[DIM+1], str_tmp1[DIM+1], str_tmp2[DIM+1];
char *re, *re_rv;

 for (i=0;i<=DIM-1;i++) {
   str1[i]=atoc[*(s2+i)];
   str2[i]=atoc[*(d2+i)];
 }
 i=0;
 do {
    strxfrm(str_tmp1,str1+i,DIM-i);

   for (j=0;j<=DIM-i-1;j++) {
      strxfrm_mod(str_tmp2,str_tmp1,DIM-i-j);
     re=strstr(str2,str_tmp2);
     re_rv=strstr_rv(str2, str_tmp2);
    if (re==NULL);
    else {
       match=strlen(str_tmp2);
       dist= abs(DIM-strlen(re)-i) > abs(DIM-strlen(re_rv)-i) ?
           abs(DIM-strlen(re)-i) : abs(DIM-strlen(re_rv)-i);
      if (match<len1) ;
      else if (match>len1) {
        len1=match;
        len2=dist;
      }
      else
        if (dist<=len2) ;
        else len2=dist;
    }
   }
  i++;
  } while (i<=DIM-1 && strlen(str_tmp1)>=len1);

  *pds=len2;
  return (len1);
}

int *rightneigh(int *s3,int i)
```

```
{
int j;

 for (j=0;j<=DIM-1;j++) {
    if (j==0) neighbor[j]=i;
    else neighbor[j]= *(s3-1+j);
 }
  return (neighbor);
}

int *leftneigh(int *s4, int i)
{
int j;
 for (j=0;j<=DIM-1;j++) {
    if (j==(DIM-1)) neighbor[j]=i;
    else neighbor[j]= *(s4+j+1);
 }
  return (neighbor);
}

void strxfrm_mod(char *str1, char *str2, int count)
{
int i;
  for (i=0;i<count;i++)
     *(str1+i)= *(str2+i);
  *(str1+count)='\0';
}

void record(int *n1, int *n2)
{
   ++load[conv(n1)][conv(n2)];
}

int conv(int *n)
{
register int i;
long sum;

  for (sum=0, i=0; i<=DIM-1; i++)
     sum += (*(n+DIM-i-1))*(long)pow((double)dlt,(double)i);
  return (sum);
}

/* Function "strstr_rv" returns a pointer to the last occurrence of the string pointed to by str2 in the string pointed
to by str1. It returns a null pointer if no match is found.        */
char *strstr_rv(char *str1, char *str2)
{
char *Ptr, *TmpPtr;

   Ptr = strstr(str1, str2);
  if (Ptr==NULL) return (Ptr);
  else
     while (strlen(Ptr)>=strlen(str2)) {
       TmpPtr = strstr(Ptr+1, str2);
      if (TmpPtr!=NULL) Ptr = TmpPtr;
      else return (Ptr);
     }
  return (Ptr);
 }
```

```
/* The following program can be used to derive the delay-thoughput performance of the one-level BdBG networks
using the "RFR" routing algorithm.

                Program name: DL_RFR.
                By Zhi Feng.
*/

#define NoK 12          /* Number of points of the normailized offered loads. */
#define DIM 6
#define NODE 1024

#include "stdlib.h"
#include "math.h"
#include "string.h"

int dlt, next[DIM];
int *left(int *n1, int *n2), *right(int *n1, int *n2);
int mergefwd(int *s, int *d), mergebwd(int *s, int *d);
long load[NODE][NODE];
void strxfrm_mod(char *str1, char *str2, int count);
void check(int *ss, int *dd), record(int *n1, int *n2);
void send_fwd(int *sc, int *de), send_bwd(int *sc, int *de);

char atoc[]="0123456789";

/* The only difference of this program from the program MPL_RFR is that this one has the calculation of normal-
ized delay according to the traffic load at each link. */
void main(argc,argv)
int argc;
char *argv[];
{
register int s[DIM], d[DIM];
int i, j, k;
long max=0, sum=0, node;
double delay,lam[NoK], mpath;

   dlt=atoi(argv[1]);
   for (k=0; k<=NoK-1; k++) lam[k] = atof(argv[k+2]);
   printf("delta=%d dimension=%d\n",dlt,DIM);
   node = pow((double)dlt,(double)DIM);


   for (s[0]=0;s[0]<=dlt-1;s[0]++)
   for (s[1]=0;s[1]<=dlt-1;s[1]++)
   for (s[2]=0;s[2]<=dlt-1;s[2]++)
   for (s[3]=0;s[3]<=dlt-1;s[3]++)
   for (s[4]=0;s[4]<=dlt-1;s[4]++)
   for (s[5]=0;s[5]<=dlt-1;s[5]++)
   for (d[0]=0;d[0]<=dlt-1;d[0]++)
   for (d[1]=0;d[1]<=dlt-1;d[1]++)
   for (d[2]=0;d[2]<=dlt-1;d[2]++)
   for (d[3]=0;d[3]<=dlt-1;d[3]++)
   for (d[4]=0;d[4]<=dlt-1;d[4]++)
   for (d[5]=0;d[5]<=dlt-1;d[5]++)

     check(s,d);

   for (i=0; i<=node-1; i++)
```

```
 for (j=0; j<=node-1; j++) {
   if (max<load[i][j])
     max = load[i][j];
     sum += load[i][j];
 }

  mpath=(float)sum/(node*(node-1.));
  printf("maximun edge loading=%d  sum=%d\n",max, sum);
   printf("mpath=%f\n",mpath);

  mpath=(float)sum/(node*(node-1.));
  printf("maximun edge loading=%d\n",max);
   printf("mpath=%f\n",mpath);

  for (k=0; k<=NoK-1; k++)  {
  delay = 0.0;

  for (i=0; i<=node-1; i++)
  for (j=0; j<=node-1; j++)
   delay+=load[i][j]/(1.-load[i][j]*lam[k]);

    delay=delay/(float)(node*(node-1.));
    printf("lam=%12.9f delay=%9.5f\n", lam[k], delay);
 }
}

void check (int *ss, int *dd)
{
int i, pathfwd, pathbwd;
char str_ss[DIM+1], str_dd[DIM+1];

  for (i=0;i<=DIM-1;i++) {
       str_ss[i]=atoc[*(ss+i)];
      str_dd[i]=atoc[*(dd+i)];
  }

  if (strcmp(str_ss, str_dd) ==  0) ;
  else {

    pathfwd = mergefwd(ss, dd);
    pathbwd = mergebwd(ss, dd);

    if (pathfwd > pathbwd) send_fwd(ss, dd);
    else if (pathfwd < pathbwd) send_bwd(ss, dd);
    else
      if ((random()&01) == 1) send_fwd(ss, dd);
      else send_bwd(ss, dd);
  }
}

int mergefwd(int *s, int *d)
{
int i;
char str_s[DIM+1], str_d[DIM+1],
     str_tmp1[DIM+1], str_tmp2[DIM+1];

   for (i=0;i<=DIM-1;i++) {
        str_s[i]=atoc[*(s+i)];
        str_d[i]=atoc[*(d+i)];
```

```
  }

  for (i=0;i<=DIM-1;i++) {
     strxfrm(str_tmp1, str_s+i, DIM-i);
     strxfrm_mod(str_tmp2, str_d, DIM-i);

     if (strcmp(str_tmp1, str_tmp2) == 0)
       return (strlen(str_tmp1));
  }
  return 0;
}

int mergebwd(int *s, int *d)
{
int i;
char str_s[DIM+1], str_d[DIM+1],
    str_tmp1[DIM+1], str_tmp2[DIM+1];

  for (i=0;i<=DIM-1;i++) {
      str_s[i]=atoc[*(s+i)];
     str_d[i]=atoc[*(d+i)];
  }

  for (i=0;i<=DIM-1;i++) {
     strxfrm_mod(str_tmp1, str_s, DIM-i);
     strxfrm(str_tmp2, str_d+i, DIM-i);

     if (strcmp(str_tmp1, str_tmp2) == 0)
       return (strlen(str_tmp1));
  }
  return 0;
}

void send_fwd(int *sc, int *de)
{
int i, *Ptrnext, Tmpst[DIM];
char str_sc[DIM+1], str_de[DIM+1];

  for (i=0;i<=DIM-1;i++) {
      str_sc[i] = atoc[*(sc+i)];
     str_de[i] = atoc[*(de+i)];
     Tmpst[i] = *(sc+i);
  }

   if (strcmp(str_sc, str_de) == 0) printf("error1");
  else {
     Ptrnext = left(Tmpst, de);
     record(Tmpst, Ptrnext);
     check(Ptrnext, de);
  }
}

void send_bwd(int *sc, int *de)
{
int i, *Ptrnext, Tmpst[DIM];
char str_sc[DIM+1], str_de[DIM+1];

    memset(str_sc,'\0',sizeof(str_sc));
    memset(str_de,'\0',sizeof(str_de));
```

```
  for (i=0;i<=DIM-1;i++) {
      str_sc[i] = atoc[*(sc+i)];
      str_de[i] = atoc[*(de+i)];
      Tmpst[i] = *(sc+i);
  }

  if (strcmp(str_sc, str_de) == 0) printf("error2");
  else {
     Ptrnext = right(Tmpst, de);
     record(Tmpst, Ptrnext);
     check(Ptrnext, de);
  }
}

int *left(int *n1, int *n2)
{
int i, path;

  path = mergefwd(n1, n2);
  for (i=0; i<=DIM-2; i++)
     next[i] = *(n1+i+1);
  next[DIM-1] = *(n2+path);
  return (next);
}

int *right(int *n1, int *n2)
{
int i, path;

  path = mergebwd(n1, n2);
  next[0] = *(n2+DIM-path-1);
  for (i=1; i<=DIM-1; i++)
     next[i] = *(n1+i-1);
  return (next);
}

void strxfrm_mod(char *str1, char *str2, int count)
{
int i;
  for (i=0;i<count;i++)
     *(str1+i)= *(str2+i);
  *(str1+count)='\0';
}

void record(int *n1, int *n2)
{
   ++load[conv(n1)][conv(n2)];
}

int conv(int *n)
{
register int i;
long sum;

  for (sum=0, i=0; i<=DIM-1; i++)
     sum += (*(n+DIM-i-1))*(long)pow((double)dlt,(double)i);
  return (sum);
}
```

B. Programs for Two-level BdBG Networks

```
/* The following program finds the optimum bridge node in terms of the node-mean-path-length of a BdBG networks using the "RFR" routing algorithm. This program can be used to construct the 2-level BdBG networks.  */

                Program name: BRIDGE_RFR.
                By Zhi Feng.
*/

#define NODE 1024
#define DIM  5

#include "stdlib.h"
#include "math.h"
#include "string.h"

int dlt, next[DIM];
int *left(int *n1, int *n2), *right(int *n1, int *n2);
int mergefwd(int *s, int *d), mergebwd(int *s, int *d);
long load[NODE][NODE];
void strxfrm_mod(char *str1, char *str2, int count);
void check(int *ss, int *dd), record(int *n1, int *n2);
void send_fwd(int *sc, int *de), send_bwd(int *sc, int *de);

char atoc[]="0123456789";


void main(argc,argv)
int argc;
char *argv[];
{
register int s[DIM], d[DIM];
int i, j, min;
long max=0, sum=0, node;
struct {
      int nd[DIM];
      int cnt;
      int summ;
      double mpath;
  } tab[NODE/2], *pt;


   dlt=atoi(argv[1]);
   printf("delta=%d dimension=%d\n",dlt,DIM);

   node = pow((double)dlt,(double)DIM);

   s[0]=0;
   for (s[1]=0;s[1]<=dlt-1;s[1]++)
   for (s[2]=0;s[2]<=dlt-1;s[2]++)
   for (s[3]=0;s[3]<=dlt-1;s[3]++)
   for (s[4]=0;s[4]<=dlt-1;s[4]++)
 /*  for (s[5]=0;s[5]<=dlt-1;s[5]++)
   for (s[6]=0;s[6]<=dlt-1;s[6]++)
   for (s[7]=0;s[7]<=dlt-1;s[7]++)
   for (s[8]=0;s[8]<=dlt-1;s[8]++)
   for (s[9]=0;s[9]<=dlt-1;s[9]++)  */  {
```

```
    for (i=0; i<=node-1; i++)
    for (j=0; j<=node-1; j++)
      load[i][j] = 0;

   sum = 0;

  for (d[0]=0;d[0]<=dlt-1;d[0]++)
  for (d[1]=0;d[1]<=dlt-1;d[1]++)
  for (d[2]=0;d[2]<=dlt-1;d[2]++)
  for (d[3]=0;d[3]<=dlt-1;d[3]++)
  for (d[4]=0;d[4]<=dlt-1;d[4]++)
/*  for (d[5]=0;d[5]<=dlt-1;d[5]++)
  for (d[6]=0;d[6]<=dlt-1;d[6]++)
  for (d[7]=0;d[7]<=dlt-1;d[7]++)
  for (d[8]=0;d[8]<=dlt-1;d[8]++)
  for (d[9]=0;d[9]<=dlt-1;d[9]++)  */

    {  check(s,d);
       check(d,s);    }

  for (i=0; i<=node-1; i++)
  for (j=0; j<=node-1; j++)
      sum += load[i][j];

    for(i=0; i<=DIM-1; i++) tab[conv(s)].nd[i] = s[i];
    ++tab[conv(s)].cnt;
    tab[conv(s)].summ = sum;
    tab[conv(s)].mpath = (double)sum/(double)(2*node);

    for (i=0; i<=node/dlt-1; i++)
      if (tab[i].summ == sum && i != conv(s))  {
          tab[i].cnt++;
          tab[conv(s)].cnt++;
      }
  }

/*    for (i=0; i<=node/dlt-1; i++)  {
       printf("Chosen node is '");
       for (j=0; j<=DIM-1; j++) printf("%d", tab[i].nd[j] );
       printf("'  ");
       printf("cnt =%d  ", tab[i].cnt);
         printf("mean path length to this node=%f\n",tab[i].mpath);
   }                                        */

      min = tab[0].summ;
      for (i=0; i<=node/dlt-1; i++)
        if (min > tab[i].summ ) {
           min = tab[i].summ ;
           pt = &tab[i];
        }

      printf("Optimum bridge is '");
      for (i=0; i<DIM; i++) printf("%d", pt->nd[i]);
      printf("'  ");
       printf("cnt =%d  ", pt->cnt);
         printf("mean path length to this node=%f\n",pt->mpath);
}
```

```
void check (int *ss, int *dd)
{
int i, pathfwd, pathbwd;
char str_ss[DIM+1], str_dd[DIM+1];

   memset(str_ss,'\0',sizeof(str_ss));
   memset(str_dd,'\0',sizeof(str_dd));

  for (i=0;i<=DIM-1;i++) {
      str_ss[i]=atoc[*(ss+i)];
     str_dd[i]=atoc[*(dd+i)];
  }

  if (strcmp(str_ss, str_dd) == 0) ;
  else {

    pathfwd = mergefwd(ss, dd);
    pathbwd = mergebwd(ss, dd);

    if (pathfwd > pathbwd) send_fwd(ss, dd);
    else if (pathfwd < pathbwd) send_bwd(ss, dd);
    else
      if ((random()&01) == 1) send_fwd(ss, dd);
      else send_bwd(ss, dd);
  }
}


int mergefwd(int *s, int *d)
{
int i;
char str_s[DIM+1], str_d[DIM+1],
    str_tmp1[DIM+1], str_tmp2[DIM+1];

   memset(str_s,'\0',sizeof(str_s));
   memset(str_d,'\0',sizeof(str_d));

  for (i=0;i<=DIM-1;i++) {
      str_s[i]=atoc[*(s+i)];
     str_d[i]=atoc[*(d+i)];
  }

  for (i=0;i<=DIM-1;i++) {
     strxfrm(str_tmp1, str_s+i, DIM-i);
     strxfrm_mod(str_tmp2, str_d, DIM-i);

     if (strcmp(str_tmp1, str_tmp2) == 0)
       return (strlen(str_tmp1));
  }
  return 0;
}

int mergebwd(int *s, int *d)
{
int i;
char str_s[DIM+1], str_d[DIM+1],
    str_tmp1[DIM+1], str_tmp2[DIM+1];

   memset(str_s,'\0',sizeof(str_s));
```

```
   memset(str_d,'\0',sizeof(str_d));

  for (i=0;i<=DIM-1;i++) {
      str_s[i]=atoc[*(s+i)];
      str_d[i]=atoc[*(d+i)];
  }


  for (i=0;i<=DIM-1;i++) {
     strxfrm_mod(str_tmp1, str_s, DIM-i);
     strxfrm(str_tmp2, str_d+i, DIM-i);

     if (strcmp(str_tmp1, str_tmp2) == 0)
       return (strlen(str_tmp1));
  }
  return 0;
}

void send_fwd(int *sc, int *de)
{
int i, *Ptrnext, Tmpst[DIM];
char str_sc[DIM+1], str_de[DIM+1];

    memset(str_sc,'\0',sizeof(str_sc));
    memset(str_de,'\0',sizeof(str_de));

  for (i=0;i<=DIM-1;i++) {
      str_sc[i] = atoc[*(sc+i)];
      str_de[i] = atoc[*(de+i)];

      Tmpst[i] = *(sc+i);
  }

   if (strcmp(str_sc, str_de) == 0) printf("error1");
  else {
     Ptrnext = left(Tmpst, de);
     record(Tmpst, Ptrnext);
     check(Ptrnext, de);
  }
}

void send_bwd(int *sc, int *de)
{
int i, *Ptrnext, Tmpst[DIM];
char str_sc[DIM+1], str_de[DIM+1];

    memset(str_sc,'\0',sizeof(str_sc));
    memset(str_de,'\0',sizeof(str_de));

  for (i=0;i<=DIM-1;i++) {
      str_sc[i] = atoc[*(sc+i)];
      str_de[i] = atoc[*(de+i)];

      Tmpst[i] = *(sc+i);
  }

   if (strcmp(str_sc, str_de) == 0) printf("error2");
  else {
     Ptrnext = right(Tmpst, de);
```

```
    record(Tmpst, Ptrnext);
    check(Ptrnext, de);
  }
}

int *left(int *n1, int *n2)
{
int i, path;

  path = mergefwd(n1, n2);

  for (i=0; i<=DIM-2; i++)
    next[i] = *(n1+i+1);

  next[DIM-1] = *(n2+path);

  return (next);
}

int *right(int *n1, int *n2)
{
int i, path;

  path = mergebwd(n1, n2);
  next[0] = *(n2+DIM-path-1);
  for (i=1; i<=DIM-1; i++)
    next[i] = *(n1+i-1);
  return (next);
}

/* Function "strxfrm_mod" transforms the first "count" characters of the string pointed to by str2 then puts the
result into the string pointed to by str1 which ended with a null pointer. */
void strxfrm_mod(char *str1, char *str2, int count)
{
int i;
  for (i=0;i<count;i++)
    *(str1+i)= *(str2+i);
  *(str1+count)='\0';
}

/* Function "record" records the traffic loads between the nodes
  pointed to by n1 and n2.                                    */

void record(int *n1, int *n2)
{
  ++load[conv(n1)][conv(n2)];
}

/* Function "conv" converts the digit pointed to by n into a decimal digit and then return it.                    */
int conv(int *n)
{
register int i;
long sum;

  for (sum=0, i=0; i<=DIM-1; i++)
    sum += (*(n+DIM-i-1))*(long)pow((double)dlt,(double)i);
  return (sum);
}
```

```
/* The following program can be used to derive the delay-thoughput performance of the local cluster in the 2-level
BdBG networks using the "RFR" routing algorithm. This program must be combined with Program DL_RFR_L2
which is listed later to get the performance of the 2-level BdBG network.      */

                    Program name: DL_RFR_L1.
                    By Zhi Feng.
*/

#define DIM  4
#define NoK 7
#define NODE 1024

#include "stdlib.h"
#include "math.h"
#include "string.h"

int dlt, n_node, n_cluster, next[DIM];
double alpha;                                   */ The locality factor.  */
int *left(int *n1, int *n2), *right(int *n1, int *n2);
int mergefwd(int *s, int *d), mergebwd(int *s, int *d);
float load[NODE][NODE];
void strxfrm_mod(char *str1, char *str2, int count);
void check(int *ss, int *dd),
    record_lo(int *n1, int *n2), record_re(int *n1, int *n2);
void send_fwd(int *sc, int *de), send_bwd(int *sc, int *de);
char str_source[DIM+1];

char atoc[]="0123456789";
char *str_bridge = "0110";              /* The bridge address  */

/* This program is similar to Program DL_RFR except that the locality factor is considered and the edge loads de-
pend also on the number of clusters the network has.  */
void main(argc,argv)
int argc;
char *argv[];
{
register int s[DIM], d[DIM];
int i, j, k;
double max=0, sum=0;
double lam[NoK], delay=0.0, Tdelay;

   dlt=atoi(argv[1]);
   n_cluster = atoi(argv[2]);
   alpha = atof(argv[3]);
   for (k=0; k<=NoK-1; k++) lam[k] = atof(argv[k+4]);
   printf("Level 1 data results:\n");
   printf("delta=%d dimension=%d n_cluster=%d\n",dlt,DIM,n_cluster);
   printf("Chosen bridge is ' %s ' \n", str_bridge);
     n_node = pow((double)dlt,(double)DIM);
   printf("Total # of links =%d\n",dlt*n_node-dlt);
   printf("alpha = %f\n",alpha);


   for (s[0]=0;s[0]<=dlt-1;s[0]++)
   for (s[1]=0;s[1]<=dlt-1;s[1]++)
   for (s[2]=0;s[2]<=dlt-1;s[2]++)
   for (s[3]=0;s[3]<=dlt-1;s[3]++)
 /*  for (s[4]=0;s[4]<=dlt-1;s[4]++)
   for (s[5]=0;s[5]<=dlt-1;s[5]++)
```

```
 for (s[6]=0;s[6]<=dlt-1;s[6]++)
 for (s[7]=0;s[7]<=dlt-1;s[7]++)
 for (s[8]=0;s[8]<=dlt-1;s[8]++)
 for (s[9]=0;s[9]<=dlt-1;s[9]++)  */
 for (d[0]=0;d[0]<=dlt-1;d[0]++)
 for (d[1]=0;d[1]<=dlt-1;d[1]++)
 for (d[2]=0;d[2]<=dlt-1;d[2]++)
 for (d[3]=0;d[3]<=dlt-1;d[3]++)
/*  for (d[4]=0;d[4]<=dlt-1;d[4]++)
 for (d[5]=0;d[5]<=dlt-1;d[5]++)
 for (d[6]=0;d[6]<=dlt-1;d[6]++)
 for (d[7]=0;d[7]<=dlt-1;d[7]++)
 for (d[8]=0;d[8]<=dlt-1;d[8]++)
 for (d[9]=0;d[9]<=dlt-1;d[9]++)  */ {

    for (i=0;i<=DIM-1;i++)   str_source[i] = atoc[*(s+i)];
    check(s,d);
 }


 for (k=0; k<=NoK-1; k++)  {
   sum = 0.0;
   delay=0.0;

 for (i=0; i<=n_node-1; i++)
 for (j=0; j<=n_node-1; j++)  {
   if  (max<load[i][j])
     max = load[i][j];
     sum += load[i][j];
        delay+=load[i][j]/(1.-load[i][j]*lam[k]);
 }

    if (k==0)
       printf("maximun edge loading=%f  sum=%f\n",max, sum);

    if (lam[k] <= 1.0/max)  {
      Tdelay = delay / (double)(n_node*(n_node*n_cluster-1)) ;
      printf("lambda=%12.9f   L1_delay=%9.5f\n", lam[k], Tdelay);
  }
 }
}


void check (int *ss, int *dd)
{
int i, pathfwd, pathbwd;
char str_ss[DIM+1], str_dd[DIM+1];

   memset(str_ss,'\0',sizeof(str_ss));
   memset(str_dd,'\0',sizeof(str_dd));

  for (i=0;i<=DIM-1;i++)  {
      str_ss[i]=atoc[*(ss+i)];
     str_dd[i]=atoc[*(dd+i)];
  }

  if (strcmp(str_ss, str_dd) ==  0) ;
  else  {
```

```
    pathfwd = mergefwd(ss, dd);
    pathbwd = mergebwd(ss, dd);

    if (pathfwd > pathbwd) send_fwd(ss, dd);
    else if (pathfwd < pathbwd) send_bwd(ss, dd);
    else
      if ((random()&01) == 0) send_fwd(ss, dd);
      else send_bwd(ss, dd);
  }
}

int mergefwd(int *s, int *d)
{
int i;
char str_s[DIM+1], str_d[DIM+1],
     str_tmp1[DIM+1], str_tmp2[DIM+1];

   memset(str_s,'\0',sizeof(str_s));
   memset(str_d,'\0',sizeof(str_d));

  for (i=0;i<=DIM-1;i++) {
      str_s[i]=atoc[*(s+i)];
      str_d[i]=atoc[*(d+i)];
  }

  for (i=0;i<=DIM-1;i++) {
     strxfrm(str_tmp1, str_s+i, DIM-i);
     strxfrm_mod(str_tmp2, str_d, DIM-i);

     if (strcmp(str_tmp1, str_tmp2) == 0)
       return (strlen(str_tmp1));
  }
  return 0;
}

int mergebwd(int *s, int *d)
{
int i;
char str_s[DIM+1], str_d[DIM+1],
     str_tmp1[DIM+1], str_tmp2[DIM+1];

   memset(str_s,'\0',sizeof(str_s));
   memset(str_d,'\0',sizeof(str_d));

  for (i=0;i<=DIM-1;i++) {
      str_s[i]=atoc[*(s+i)];
      str_d[i]=atoc[*(d+i)];
  }


  for (i=0;i<=DIM-1;i++) {
     strxfrm_mod(str_tmp1, str_s, DIM-i);
     strxfrm(str_tmp2, str_d+i, DIM-i);

     if (strcmp(str_tmp1, str_tmp2) == 0)
       return (strlen(str_tmp1));
  }
  return 0;
}
```

```
void send_fwd(int *sc, int *de)
{
int i, *Ptrnext, Tmpst[DIM];
char str_sc[DIM+1], str_de[DIM+1];

   memset(str_sc,'\0',sizeof(str_sc));
   memset(str_de,'\0',sizeof(str_de));

  for (i=0;i<=DIM-1;i++) {
      str_sc[i] = atoc[*(sc+i)];
      str_de[i] = atoc[*(de+i)];
      Tmpst[i] = *(sc+i);
  }

  if (strcmp(str_sc, str_de) == 0) printf("error1");
  else {
     Ptrnext = left(Tmpst, de);

     if (strcmp(str_source, str_bridge)==0 || strcmp(str_de, str_bridge)==0)
        record_re(Tmpst, Ptrnext);              /* To record the remote traffic  */
     record_lo(Tmpst, Ptrnext);                 /* To record the local traffic   */
     check(Ptrnext, de);
  }
}

void send_bwd(int *sc, int *de)
{
int i, *Ptrnext, Tmpst[DIM];
char str_sc[DIM+1], str_de[DIM+1];

   memset(str_sc,'\0',sizeof(str_sc));
   memset(str_de,'\0',sizeof(str_de));

  for (i=0;i<=DIM-1;i++) {
      str_sc[i] = atoc[*(sc+i)];
      str_de[i] = atoc[*(de+i)];

      Tmpst[i] = *(sc+i);
  }

  if (strcmp(str_sc, str_de) == 0) printf("error2");
  else {
     Ptrnext = right(Tmpst, de);

     if (strcmp(str_sc, str_bridge)==0 || strcmp(str_de,str_bridge)==0)
        record_re(Tmpst, Ptrnext);
     record_lo(Tmpst, Ptrnext);
     check(Ptrnext, de);
  }
}

int *left(int *n1, int *n2)
{
int i, path;

  path = mergefwd(n1, n2);

  for (i=0; i<=DIM-2; i++)
```

```
    next[i] = *(n1+i+1);

  next[DIM-1] = *(n2+path);

  return (next);
}

int *right(int *n1, int *n2)
{
int i, path;

  path = mergebwd(n1, n2);

  next[0] = *(n2+DIM-path-1);

  for (i=1; i<=DIM-1; i++)
    next[i] = *(n1+i-1);

  return (next);
}

void strxfrm_mod(char *str1, char *str2, int count)
{
int i;
  for (i=0;i<count;i++)
    *(str1+i)= *(str2+i);

  *(str1+count)='\0';
}

/* The record_lo function records the local traffic according to the locality factor. */
void record_lo(int *n1, int *n2)
{
  load[conv(n1)][conv(n2)] +=
    alpha*(double)(n_cluster*n_node-1)/(double)(n_node-1);
}

/* The record_lo function records the remote traffic according to the locality factor. */
void record_re(int *n1, int *n2)
{
  if (n_cluster != 1)  {
  load[conv(n1)][conv(n2)] +=
 (1.-alpha)*(double)(n_cluster*n_node-1)/(double)((n_cluster-1)*n_node);

  load[conv(n2)][conv(n1)] +=
(1.-alpha)*(double)(n_cluster*n_node-1);
 }
}


int conv(int *n)
{
register int i;
long sum;

  for (sum=0, i=0; i<=DIM-1; i++)
    sum += (*(n+DIM-i-1))*(long)pow((double)dlt,(double)i);
  return (sum);
}
```

```
/* The following program can be used to derive the delay-thoughput performance of the upper level in the 2-level
BdBG networks using the "RFR" routing algorithm. This program must be combined with Program DL_RFR_L1
which is listed earlier to get the performance of the 2-level BdBG network.      */

                    Program name: DL_RFR_L2.
                    By Zhi Feng.
*/

#define DIM  8
#define NoK 7
#define NODE 1024

#include "stdlib.h"
#include "math.h"
#include "string.h"

int dlt, n_node, next[DIM];
float alpha;
int *left(int *n1, int *n2), *right(int *n1, int *n2), mergefwd(int *s, int *d), mergebwd(int *s, int *d);
int load[NODE][NODE];
void strxfrm_mod(char *str1, char *str2, int count), check(int *ss, int *dd), record(int *n1, int *n2);
void send_fwd(int *sc, int *de), send_bwd(int *sc, int *de);

char atoc[]="0123456789";

void main(argc,argv)
int argc;
char *argv[];
{
register int s[DIM], d[DIM];
int i, j, k, node;
double max=0., sum=0., tem;
double lam[NoK], delay=0.0;

   dlt=atoi(argv[1]);
   n_node=atoi(argv[2]);
   alpha = atof(argv[3]);
   for (k=0; k<=NoK-1; k++) lam[k] = atof(argv[k+4]);
   printf("Level 2--delta=%d dimension=%d node/cluster =%d\n",dlt,DIM,n_node);
  node = pow((double)dlt,(double)DIM);

   for (s[0]=0;s[0]<=dlt-1;s[0]++)
   for (s[1]=0;s[1]<=dlt-1;s[1]++)
   for (s[2]=0;s[2]<=dlt-1;s[2]++)
   for (s[3]=0;s[3]<=dlt-1;s[3]++)
   for (s[4]=0;s[4]<=dlt-1;s[4]++)
   for (s[5]=0;s[5]<=dlt-1;s[5]++)
   for (s[6]=0;s[6]<=dlt-1;s[6]++)
   for (s[7]=0;s[7]<=dlt-1;s[7]++)
   for (d[0]=0;d[0]<=dlt-1;d[0]++)
   for (d[1]=0;d[1]<=dlt-1;d[1]++)
   for (d[2]=0;d[2]<=dlt-1;d[2]++)
   for (d[3]=0;d[3]<=dlt-1;d[3]++)
   for (d[4]=0;d[4]<=dlt-1;d[4]++)
   for (d[5]=0;d[5]<=dlt-1;d[5]++)
   for (d[6]=0;d[6]<=dlt-1;d[6]++)
   for (d[7]=0;d[7]<=dlt-1;d[7]++)

     check(s,d);
```

```
 for (k=0; k<=NoK-1; k++)  {
   delay=0.0;
   sum=0.;

   for (i=0; i<=node-1; i++)
   for (j=0; j<=node-1; j++)  {
      tem = (double)load[i][j]*(1.0-alpha)*
          ((double)(n_node*(n_node*node-1))/(double)(node-1));
     if  (max<tem)  max = tem;
     sum += tem;
      delay+=tem/(1.-(double)tem*lam[k]);
   }

   if (k==0)  {
     printf("       maximun edge loading=%f  ",max);
     printf("sum=%f\n",sum);
   }

  if (lam[k] <= 1.0/max)  {
    delay = delay / (double)(node*n_node*(n_node*node-1)) ;
    printf("lambda=%12.9f   L2_delay=%9.5f\n", lam[k], delay);
  }
 }
}

void check (int *ss, int *dd)
{
int i, pathfwd, pathbwd;
char str_ss[DIM+1], str_dd[DIM+1];

  for (i=0;i<=DIM-1;i++)  {
      str_ss[i]=atoc[*(ss+i)];
     str_dd[i]=atoc[*(dd+i)];
  }

  if (strcmp(str_ss, str_dd) ==  0) ;
  else  {

    pathfwd = mergefwd(ss, dd);
    pathbwd = mergebwd(ss, dd);

    if (pathfwd > pathbwd) send_fwd(ss, dd);
    else if (pathfwd < pathbwd) send_bwd(ss, dd);
    else
      if ((random()&01) == 1) send_fwd(ss, dd);
      else send_bwd(ss, dd);
  }
}

int mergefwd(int *s, int *d)
{
int i;
char str_s[DIM+1], str_d[DIM+1],
    str_tmp1[DIM+1], str_tmp2[DIM+1];

  for (i=0;i<=DIM-1;i++)  {
      str_s[i]=atoc[*(s+i)];
     str_d[i]=atoc[*(d+i)];
```

```
  }
  for (i=0;i<=DIM-1;i++) {
     strxfrm(str_tmp1, str_s+i, DIM-i);
     strxfrm_mod(str_tmp2, str_d, DIM-i);
     if (strcmp(str_tmp1, str_tmp2) == 0)
       return (strlen(str_tmp1));
  }
  return 0;
}

int mergebwd(int *s, int *d)
{
int i;
char str_s[DIM+1], str_d[DIM+1],
    str_tmp1[DIM+1], str_tmp2[DIM+1];

  for (i=0;i<=DIM-1;i++) {
      str_s[i]=atoc[*(s+i)];
      str_d[i]=atoc[*(d+i)];
  }

  for (i=0;i<=DIM-1;i++) {
     strxfrm_mod(str_tmp1, str_s, DIM-i);
     strxfrm(str_tmp2, str_d+i, DIM-i);
     if (strcmp(str_tmp1, str_tmp2) == 0)
       return (strlen(str_tmp1));
  }
  return 0;
}

void send_fwd(int *sc, int *de)
{
int i, *Ptrnext, Tmpst[DIM];
char str_sc[DIM+1], str_de[DIM+1];

  for (i=0;i<=DIM-1;i++) {
      str_sc[i] = atoc[*(sc+i)];
      str_de[i] = atoc[*(de+i)];

      Tmpst[i] = *(sc+i);
  }

  if (strcmp(str_sc, str_de) == 0) printf("error1");
  else {
    Ptrnext = left(Tmpst, de);
    record(Tmpst, Ptrnext);
    check(Ptrnext, de);
  }
}

void send_bwd(int *sc, int *de)
{
int i, *Ptrnext, Tmpst[DIM];
char str_sc[DIM+1], str_de[DIM+1];

  for (i=0;i<=DIM-1;i++) {
      str_sc[i] = atoc[*(sc+i)];
      str_de[i] = atoc[*(de+i)];
```

```
    Tmpst[i] = *(sc+i);
  }

  if (strcmp(str_sc, str_de) == 0) printf("error2");
  else {
    Ptrnext = right(Tmpst, de);
    record(Tmpst, Ptrnext);
    check(Ptrnext, de);
  }
}

int *left(int *n1, int *n2)
{
int i, path;

  path = mergefwd(n1, n2);
  for (i=0; i<=DIM-2; i++)
    next[i] = *(n1+i+1);

  next[DIM-1] = *(n2+path);
  return (next);
}

int *right(int *n1, int *n2)
{
int i, path;

  path = mergebwd(n1, n2);
  next[0] = *(n2+DIM-path-1);

  for (i=1; i<=DIM-1; i++)
    next[i] = *(n1+i-1);
  return (next);
}

void strxfrm_mod(char *str1, char *str2, int count)
{
int i;
  for (i=0;i<count;i++)
    *(str1+i)= *(str2+i);
  *(str1+count)='\0';
}

void record(int *n1, int *n2)
{
  ++load[conv(n1)][conv(n2)];
}

int conv(int *n)
{
register int i;
long sum;

  for (sum=0, i=0; i<=DIM-1; i++)
    sum += (*(n+DIM-i-1))*(long)pow((double)dlt,(double)i);
  return (sum);
}
```

C. Programs for UdBG networks

```
/* The following program calculates the Mean Path Length of the Uidirectional de Bruijn Graph

                Program name: MPL_UdBG.                                    Oct.15, 1993.
                By Zhi Feng.
*/

#define DIM 6
#define NODE 1025

#include "stdlib.h"
#include "math.h"
#include "string.h"

int dlt, next[DIM];
int *left(int *n1, int *n2);
int mergefwd(int *s, int *d);
long load[NODE][NODE];
void strxfrm_mod(char *str1, char *str2, int count);
void check(int *ss, int *dd), record(int *n1, int *n2);

char atoc[]="0123456789";


main(argc,argv)
int argc;
char *argv[];
{
register int s[DIM], d[DIM];
int i, j;
long max=0, sum=0, node;
double mpath;

    dlt=atoi(argv[1]);
    printf("delta=%d dimension=%d\n",dlt,DIM);
   node = pow((double)dlt,(double)DIM);
   printf("Total # of links =%d\n",dlt*node-dlt);

   for (s[0]=0;s[0]<=dlt-1;s[0]++)
   for (s[1]=0;s[1]<=dlt-1;s[1]++)
   for (s[2]=0;s[2]<=dlt-1;s[2]++)
   for (s[3]=0;s[3]<=dlt-1;s[3]++)
   for (s[4]=0;s[4]<=dlt-1;s[4]++)
   for (s[5]=0;s[5]<=dlt-1;s[5]++)
 /*  for (s[6]=0;s[6]<=dlt-1;s[6]++)
   for (s[7]=0;s[7]<=dlt-1;s[7]++)
   for (s[8]=0;s[8]<=dlt-1;s[8]++)
   for (s[9]=0;s[9]<=dlt-1;s[9]++)  */
   for (d[0]=0;d[0]<=dlt-1;d[0]++)
   for (d[1]=0;d[1]<=dlt-1;d[1]++)
   for (d[2]=0;d[2]<=dlt-1;d[2]++)
   for (d[3]=0;d[3]<=dlt-1;d[3]++)
   for (d[4]=0;d[4]<=dlt-1;d[4]++)
   for (d[5]=0;d[5]<=dlt-1;d[5]++)
 /*  for (d[6]=0;d[6]<=dlt-1;d[6]++)
   for (d[7]=0;d[7]<=dlt-1;d[7]++)
   for (d[8]=0;d[8]<=dlt-1;d[8]++)
   for (d[9]=0;d[9]<=dlt-1;d[9]++)  */
```

```
   check(s,d);

    for (i=0; i<=node-1; i++)
    for (j=0; j<=node-1; j++) {
      if (max<load[i][j])
        max = load[i][j];
      sum += load[i][j];
    }

   mpath=(float)sum/(node*(node-1.));
  printf("maximun edge loading=%d  sum=%d   ",max, sum);
   printf("mpath=%f\n",mpath);
}


void check (int *ss, int *dd)
{
int i, *Ptrnext, Tmpst[DIM];
char str_ss[DIM+1], str_dd[DIM+1];

    memset(str_ss,'\0',sizeof(str_ss));
   memset(str_dd,'\0',sizeof(str_dd));

  for (i=0;i<=DIM-1;i++) {
       str_ss[i]=atoc[*(ss+i)];
      str_dd[i]=atoc[*(dd+i)];

       Tmpst[i] = *(ss+i);
  }

  if (strcmp(str_ss, str_dd) == 0) ;
  else {

     Ptrnext = left(Tmpst, dd);
     record(Tmpst, Ptrnext);
     check(Ptrnext, dd);
   }
 }


int *left(int *n1, int *n2)
{
int i, path;

   path = mergefwd(n1, n2);

   for (i=0; i<=DIM-2; i++)
     next[i] = *(n1+i+1);

   next[DIM-1] = *(n2+path);

   return (next);
}

int mergefwd(int *s, int *d)
{
int i;
char str_s[DIM+1], str_d[DIM+1],
```

```
   str_tmp1[DIM+1], str_tmp2[DIM+1];

  memset(str_s,'\0',sizeof(str_s));
  memset(str_d,'\0',sizeof(str_d));

 for (i=0;i<=DIM-1;i++) {
     str_s[i]=atoc[*(s+i)];
     str_d[i]=atoc[*(d+i)];
 }

 for (i=0;i<=DIM-1;i++) {
    strxfrm(str_tmp1, str_s+i, DIM-i);
    strxfrm_mod(str_tmp2, str_d, DIM-i);

    if (strcmp(str_tmp1, str_tmp2) == 0)
      return (strlen(str_tmp1));
 }
  return 0;
}

void strxfrm_mod(char *str1, char *str2, int count)
{
int i;
   for (i=0;i<count;i++)
     *(str1+i)= *(str2+i);

   *(str1+count)='\0';
}


/* Function "record" records the traffic loads between the nodes
   pointed to by n1 and n2.                                    */

void record(int *n1, int *n2)
{
   ++load[conv(n1)][conv(n2)];
}

/* Function "conv" converts the digit pointed to by n into
   a decimal digit and then return it.                         */

int conv(int *n)
{
register int i;
long sum;

   for (sum=0, i=0; i<=DIM-1; i++)
     sum += (*(n+DIM-i-1))*(long)pow((double)dlt,(double)i);

   return (sum);
 }
```

```
/* The following program finds the optimum bridge node in terms of the node-mean-path-length of a UdBG net-
works. This program can be used to construct the 2-level UdBG networks.      */

                Program name: BRIDGE_UdBG.                                  Nov.15, 1993
                By Zhi Feng.
*/
#define DIM 6
#define NODE 1024

#include "stdlib.h"
#include "math.h"
#include "string.h"

int dlt, next[DIM], *left(int *n1, int *n2), mergefwd(int *s, int *d);
long load[NODE][NODE];
void strxfrm_mod(char *str1, char *str2, int count), check(int *ss, int *dd), record(int *n1, int *n2);
char atoc[]="0123456789";

main(argc,argv)
int argc;
char *argv[];
{
register int s[DIM], d[DIM];
int i, j,min;
long  sum=0,max=0, node;
struct {
          int nd[DIM];
          int cnt;
          int summ;
          double mpath;
  } tab[NODE/2], *pt;

   dlt=atoi(argv[1]);
   printf("delta=%d dimension=%d\n",dlt,DIM);

    node = pow((double)dlt,(double)DIM);

  s[0] = 0;
   for (s[1]=0;s[1]<=dlt-1;s[1]++)
   for (s[2]=0;s[2]<=dlt-1;s[2]++)
   for (s[3]=0;s[3]<=dlt-1;s[3]++)
   for (s[4]=0;s[4]<=dlt-1;s[4]++)
   for (s[5]=0;s[5]<=dlt-1;s[5]++)
 /*  for (s[6]=0;s[6]<=dlt-1;s[6]++)
   for (s[7]=0;s[7]<=dlt-1;s[7]++)
   for (s[8]=0;s[8]<=dlt-1;s[8]++)
   for (s[9]=0;s[9]<=dlt-1;s[9]++)  */  {

     for (i=0; i<=node-1; i++)
     for (j=0; j<=node-1; j++)
       load[i][j] = 0;
    sum = 0;

   for (d[0]=0;d[0]<=dlt-1;d[0]++)
   for (d[1]=0;d[1]<=dlt-1;d[1]++)
   for (d[2]=0;d[2]<=dlt-1;d[2]++)
   for (d[3]=0;d[3]<=dlt-1;d[3]++)
   for (d[4]=0;d[4]<=dlt-1;d[4]++)
   for (d[5]=0;d[5]<=dlt-1;d[5]++)
```

```
/*  for (d[6]=0;d[6]<=dlt-1;d[6]++)
  for (d[7]=0;d[7]<=dlt-1;d[7]++)
  for (d[8]=0;d[8]<=dlt-1;d[8]++)
  for (d[9]=0;d[9]<=dlt-1;d[9]++)  */

  {  check(s,d);
     check(d,s);  }

    for (i=0; i<=node-1; i++)
    for (j=0; j<=node-1; j++)
      sum += load[i][j];

    for(i=0; i<=DIM-1; i++) tab[conv(s)].nd[i] = s[i];
    ++tab[conv(s)].cnt;
    tab[conv(s)].summ = sum;
    tab[conv(s)].mpath = (double)sum/(double)(2*node);

    for (i=0; i<=node/dlt-1; i++)
           if (tab[i].summ == sum && i != conv(s))  {
              tab[i].cnt++;
              tab[conv(s)].cnt++;
     }
  }

    for (i=0; i<=node/dlt-1; i++)  {
       printf("Chosen node is '");
       for (j=0; j<=DIM-1; j++) printf("%d", tab[i].nd[j] );
       printf("'  ");
       printf("cnt =%d  ", tab[i].cnt);
        printf("mean path length to this node=%f\n",tab[i].mpath);
    }

      min = tab[0].summ;
      for (i=0; i<=node/dlt-1; i++)
        if (min > tab[i].summ ) {
          min = tab[i].summ ;
          pt = &tab[i];
        }

      printf("Optimum bridge is '");
      for (i=0; i<DIM; i++) printf("%d", pt->nd[i]);
      printf("'  ");
       printf("cnt =%d  ", pt->cnt);
        printf("mean path length to this node=%f\n",pt->mpath);
}

void check (int *ss, int *dd)
{
int i, *Ptrnext, Tmpst[DIM];
char str_ss[DIM+1], str_dd[DIM+1];

  for (i=0;i<=DIM-1;i++)  {
       str_ss[i]=atoc[*(ss+i)];
      str_dd[i]=atoc[*(dd+i)];
      Tmpst[i] = *(ss+i);
  }

  if (strcmp(str_ss, str_dd) ==  0) ;
  else {
```

```
    Ptrnext = left(Tmpst, dd);
    record(Tmpst, Ptrnext);
    check(Ptrnext, dd);
  }
}

int *left(int *n1, int *n2)
{
int i, path;

  path = mergefwd(n1, n2);
  for (i=0; i<=DIM-2; i++)
    next[i] = *(n1+i+1);
  next[DIM-1] = *(n2+path);
  return (next);
}

int mergefwd(int *s, int *d)
{
int i;
char str_s[DIM+1], str_d[DIM+1],
    str_tmp1[DIM+1], str_tmp2[DIM+1];
  for (i=0;i<=DIM-1;i++) {
      str_s[i]=atoc[*(s+i)];
      str_d[i]=atoc[*(d+i)];
  }
  for (i=0;i<=DIM-1;i++) {
     strxfrm(str_tmp1, str_s+i, DIM-i);
     strxfrm_mod(str_tmp2, str_d, DIM-i);
     if (strcmp(str_tmp1, str_tmp2) == 0)
       return (strlen(str_tmp1));
  }
  return 0;
}

void strxfrm_mod(char *str1, char *str2, int count)
{
int i;
  for (i=0;i<count;i++)
    *(str1+i)= *(str2+i);
  *(str1+count)='\0';
}

void record(int *n1, int *n2)
{
  ++load[conv(n1)][conv(n2)];
}

int conv(int *n)
{
register int i;
long sum;

  for (sum=0, i=0; i<=DIM-1; i++)
    sum += (*(n+DIM-i-1))*(long)pow((double)dlt,(double)i);
  return (sum);
}
```