PERFORMANCE EVALUATION OF INTERCONNECTION NETWORKS FOR ISDN SWITCHING APPLICATIONS

by

Cheng-Leo George Lin

A Thesis Submitted to the Faculty of the

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

In Partial Fulfilment of the Requirements
For the Degree of

MASTER OF SCIENCE

WITH A MAJOR IN ELECTRICAL ENGINEERING

In the Graduate College

THE UNIVERSITY OF ARIZONA

1990

STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfilment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED:______________________________

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

____
William H. Sanders
Assistant Professor of
Electrical and Computer Engineering

____
Date

To my wife, Donna

ACKNOWLEDGMENTS

I would first like to acknowledge Dr. William H. Sanders, my thesis advisor. He gave me guidance, support and encouragement along the way. Thanks also go to Dr. Max Liu for his suggestion on the switch design and inspiring discussions. I am grateful to Dr. Ralph Martinez for his reviewing of this thesis.

Members in the Computer Center where I worked for two and half years were always very helpful and understanding. I wish to express my thanks to them here.

Finally, I would like to thank my wife, my family and friends for their moral support during this period.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Interconnection networks of various designs have been proposed for use as fast packet switches for broadband ISDN applications. While each of these designs has its own merits and drawbacks, it is not clear how they perform, relative to one another, in this application area. We use stochastic activity networks to model and simulate these designs, and show that the choice of network design to use depending on both the workload experienced and the action taken when contention for a particular switch element occurs. In particular, we use stochastic activity networks to compare three different switch designs (basic banyan, modified delta, and a design with multiplexer and demultiplexer) under both uniform and non-uniform workload assumptions, and with different resolution policies when contention for a particular switch element occurs. Regarding contention resolution, we consider two policies, one with blocking, and one where the packet is rejected and must be retransmitted. For each scenario, we determine blocking probability and mean transmission delay. We find that while traditional designs work well with uniform workloads, they do not work so well with non-uniform workloads, and, in fact, the simpler design with multiplexer and demultiplerer works better in some reject-retransmission cases. The modeified delta network, due to its multiple path, performs the best among the three designs with uniform workloads.

CHAPTER 1

INTRODUCTION

1.1 General Background

The goal of a broadband Integrated Services Digital Network (ISDN) is to integrate different types of traffic such as voice, data, image and video into an unified network. As envisioned, broadband ISDNs will be able to provide a wide variety of services in the future. With growing demands of these services, a carrier of high bandwidth and a switch capable of handling such workload are essential. Among those criteria being considered in building such network, the cost and speed of a switch are two of the most important ones. A switch not only needs to deal with high traffic intensity but also be affordable. Therefore, it is beneficial to study the performance of different switch designs.

The choice of switching method used to construct the communication network serving the various traffic types is a major design decision. Voice traffic requires low delay and high throughput, but can tolerate some errors. Interactive data traffic requires low delay and a low bit error rate, but does not typically require high throughput. Image traffic requires high throughput because of its bulk of data. It also requires a low bit error rate, but not a low delay.

There are mainly two candidates in switching techniques suitable to handle these various requirements: packet switching and circuit switching. Packet switching and circuit switching differ in many respects. The basic difference lies on that circuit switching guarantee bandwidth in advance, while packet switching does not.Thus packet switching provides a more flexible way of utilizing the resources.

However, traditional packet switching has some drawbacks. The key problems are routing and congestion control. In high-speed packet switched network, large and variable delay as well as throughput bottlenecks may develop [29]. These problems motivated the development of a "fast packet switching", which is a hardware implementation of basic switching and protocol functions. Switching is done by using large self-routing networks [30]. This fast packet switching was favored in recent study of switching techniques [13]. A class of fast packet switch designs, based on multistage interconnection networks implemented from identical switching elements, is investigated in this research. This class of networks has been considered a candidate for the packet network with high throughput because:

1. Several packets can be switched simultaneously and in parallel.
2. The switching function can be implemented in hardware.
3. This type of switches is capable of operating in either synchronous or asynchronous mode in packet level.

Multistage interconnection networks have been used extensively in multiprocessor applications and communication systems. The *banyan network* [7] is the basic type of these multistage interconnection networks.

1.2 Previous Work on Banyan-type Interconnection Networks

First proposed by Goke and Lipovski [6], the banyan network (see Figure 2.2, baseline network) has the following properties:

1. There is exactly one path from any input to any output.
2. The structure of the network is highly modularized, composed of identical switching elements with the same number of input and outputs. Hence the switch network can be easily implemented by VLSI technique.
3. The network consists of $log_b N$ stages and N/b nodes per stage, where b is the number of input for each switching element.

4. Network has the self-routing property for the packet movement from any input to any output by using a unique set of k ($k = log_b N$) digit, base b destination address.

Furthermore, banyan networks are blocking networks, i.e. packets can collide with each other and get lost in the network. There are basically three types of blocking:

1. *internal link blocking*: packets are blocked due to the contention for the same link inside the network.
2. *external blocking*, or output port blocking: packets are blocked because they are destinated to the same output port.
3. *head of line* (HOL) *blocking*: A Head Of Line blocking is caused by blocking of the packet at the head of the queue. Since the first packet is blocked, the delivery of later packets in the same queue is prohibited (blocked) even the output ports of these packet are available at that time.

The analysis of unbuffered banyan network has been studied by Patel [24] and Kumar et al. [14] In his paper, Patel analyzed the performance of unbuffered delta networks in which the interconnection pattern is a permuted form of banyan network. An uniform traffic load is applied. He indicated that the performance of the network is independent of the choice of interconnection pattern, and that the throughput of an unbuffered delta network under uniform traffic load can be expressed as a quadratic recurrence relation. Kruskal and Snir [12] provided the asymptotic solution for this recurrence relation.

To increase the performance of the banyan network and to reduce the blocking probability, several techniques have been proposed:

1. placing buffers in every switching element.
2. increasing the internal processing speed relative to the applied workload.
3. using a handshaking mechanism between stages or a back–pressure mechanism to delay the transfer of blocked packets.
4. using multiple networks in parallel to provide multiple paths from any input to any output or multiple links for each switch connection.
5. using a distribution network in front of the banyan network to distribute the load more evenly.

The performance of buffered banyan networks has been studied by several researchers [3, 10, 11, 12, 32]. In particular, an analytical packet switch model based on single-buffered banyan network was studied by Jenq [10]. The building block of this model was a 2×2 switching element. The results showed reasonably low blocking probabilities and low delays for balanced internal loads. A similar result was obtained by Dias and Jump [3]. They noticed that as the number of buffers between stages increased, the throughput converged to a constant whereas the turn-around-time increased almost linearly. They suggested the number of buffers between stages be limited to one or two. Kruskal and Snir [12] derived an equation for the performance of buffered banyan networks. Using the analysis, they compared buffered banyan networks built of different sized switches and determined where each switch size was most effective. Wu [32] studied the performance of a buffered banyan network under the mixing traffic condition. In the 4-stage single-buffered banyan network, Wu found that the overlapping point-to-point traffic had very little effect on the background uniform traffic until its throughput reached beyond 45%. If the point-to-point load increases further, the maximum throughput of the uniform traffic decreases almost linearly and become zero when the dedicated channel operated at its full speed.

Kim and Leon-Garcia [11] evaluated single-buffered and multibuffered banyan networks under nonuniform traffic patterns. Their analysis showed that the single-buffer banyan network suffered from performance degradation caused by nonuniformity of the traffic pattern. The degradation became more pronounced as the size of the network increased. The multibuffered banyan network showed an improvement in throughput capacity over the single-buffered banyan. However, the improvement was not significant when the size of the network was large.

To overcome the internal blocking problem posed by the banyan network, a front-end sorting network has been suggested [8, 9]. Packets are first sorted based on their destination addresses and then routed through the banyan network. Examples of such sort-banyan networks are: Batcher-Banyan network reported in Huang and Knauer [8], Hui and Arthurs [9]. Huang and Knauer [8] also implemented this idea in their Starlite switch. A distribution network is another option to reduce the blocking inside the banyan network. Turner's integrated services packet network [30] used a distribution network, which had the same structure as the routing network, to distribute packets evenly across all its output ports.

Finally, in his design of a fast packet switch, Newman [22, 23] adopted a distribution network, known as Benes network (see Figure 2.2), as the distribution fabric in front of the modified banyan routing network. Other approach has been taken to reduce the blocking probability.

1.3 Previous Work on Switch Designs based on Interconnection Networks

A broadband self-routing packet switch design for providing flexible multiple bit-rate broadband services was proposed by Hui and Arthurs [9]. The switch fabric delivers exactly one packet to each output port from one of the input ports which requested packet delivery to that output port. This is done by first passing the destination address through a Batcher sorting network [1], which sorts the request destinations in ascending order so that only one request for the same destination is retained. The winning request acknowledges its originating port from the output of the Batcher network, with the acknowledgement routed through a Batcher-banyan self-routing switch. The acknowledged input port then send the full packet through the same Batcher-banyan switch without any conflict. Unacknowledged ports buffer the blocked packet for reentry in the next cycle. They analyzed the throughput-delay characteristics for uniform traffic, modeled by random output port requests and a binomial distribution of packet arrival. They demonstrated with a buffer size of around 20 packets, a 50 percent loading can be achieved with almost no overflows of the buffer. They also studied the performance of the switch in the presence of periodic broadband traffic.

A feedback banyan switching network topology was described in the paper by Uematsu and Watanabe [31]. They proposed a feedback banyan switching network consisting of feedback loops to connect the output ports of the network to its input ports. The input virtual paths encounter congestion are fed back to the input ports via these loops. They are then rerouted through the switching network. This reduces congestion and realized connections with high specified throughput. However, this design needs two times as many switch cells of a normal banyan network and dismisses self-routing ability of the network.

Newman (1988) [22, 23] proposed a fast packet switch of high traffic capacity based on a nonbuffered, multistage interconnection network. Both input and output ports contain buffers. An incoming packet arrives in a first-in first-out queue. When free, the respective input port controller extracts the label from the packet at the head of queue and uses it to reference a connection table. Each input port controller operates asynchronously, at the packet level, and independently of all other controllers. The switch fabric includes a routing fabric and a distribution fabric. The routing fabric is constructed by combining self-routing, multistage interconnection networks known as delta networks. Each interconnection link in the delta network consists of two paths, a forward path to carry the data and a reverse path to carry the collision signal. The distributed fabric has a Benes topology. Its function is to distribute the incoming traffic across an delta network. Newman has examined the performance of the switch constructed by switching element of various sizes, from 2×2 to 16×16. He also investigated two algorithms, searching and flooding, in selecting between equivalent paths.

1.4 Statement of the Problem

The need of a fast and efficient switch is obvious when we march into the era of broadband ISDN. Although some work has been done to improve the performance of the banyan network, as mentioned above, a comparison of different designs is yet to be furnished. In particular, investigating the effects of various workloads and contention resolution algorithms to different switches can be important when it comes to a design decision.

Switch designs based on the banyan network are the possible solutions to the future communication needs. It is useful to compare different designs and algorithms in order to find out the most practical one in terms of the throughput and cost.

Specifically, we will compare three types of switching fabrics based on banyan networks:

1. an 8×8 modified delta network constructed from 4×4 banyan switching networks.
2. an 8×8 modified switch made of 4×4 banyan networks, multiplexers and demultiplexers (Mux-Demux).
3. an 8×8 baseline banyan network.

Switching fabric (1) has more than one path from one input port to any output port. Thus provides extra route for packets. While switching fabric (2) makes use of multiplexer and demultiplexer. Which lowers the cost of building the switch. These two switching fabrics can then be compared with (3) basic 8×8 banyan network (baseline network). More elaborated discussions will be provided in Chapter 2 and Chapter 4. I will evaluate the performance of the switching fabrics in terms of throughput, packet loss probability, and packet time delay. Since a number of paths may share common links within banyan network, it is also interesting to investigate different algorithms in handling the blocking of packets caused by simultaneous connections of more than one input and output pair. Two contention resolution algorithms will be examined, namely back-pressure and reject. The back-pressure algorithm states that whenever there is a conflict in using internal link, (i.e. a packet is blocked,) the system simply delays the transfer of that packet. Due to the holding back of one packet, there could be a series of holding back of packets in previous stages. Thus this blocking effect will migrate to the input ports of the switching fabric. The reject algorithm works as following: whenever a packet is blocked, the system rejects the packet. The rejected packet is dropped. Depending on the type of packet, it may be re-transmitted by upper layer protocol. Furthermore, applying various workloads to each switch system will allow us to better understand the types of traffic which a switch can adequately handle.

Simulation using stochastic activity networks will be used due to the complexity of the system. Stochastic activity network is an probabilistic extension of Activity Networks, which are non-deterministic models. The extension is done by specifying the spatial and temporal uncertainties with probability distributions and probability distribution functions for a subclass of activity networks that are well behaved.

Chapter 2 provides a general description of switching techniques. This starts with the comparison between circuit switching and packet switching. Then the fast packet switching is introduced in detail along with the architecture of the switch.

Chapter 3 talks about the definition of the stochastic activity network model and briefly touches the simulation tool employed in this study. It also describes the structure of the tool.

Chapter 4 states the construction of the switch models. First functional models is presented, followed by the SAN models of three types of banyan network design. Performance variables are then discussed.

Chapter 5 presents the results of the simulation runs based on constructed models.

Chapter 6 summarizes the finding of this research and suggests avenues for future study.

CHAPTER 2

OVERVIEW OF SWITCHING TECHNIQUES

2.1 Circuit versus Packet Switching

2.1.1 Traffic Characteristics

There are four types of traffic a switch should handle: voice, data, image (bulk data) and video. Each of them has different operational constraints. Voice traffic must meet the minimum requirement of sampling the speech. Furthermore, because of the nature of the voice communication, voice traffic requires small delay and high throughput. A very low error rate in this case is not a critical issue. An example is the voice conversation through current phone system. An occasional error is acceptable in these voice conversations. In contrast, data traffic can tolerate little error. Communication between terminals and host computers is a typical case of this kind of traffic. A single error in the transmitted data can change its meaning completely. In most of the cases, data traffic requires low delay (interactive). However, it does not have a particularly high throughput requirement. An high throughput is expected in image traffic because of large amount of data needed to be transmitted. It also requires a low bit error rate for the same reason as in data traffic. Low delay is not as a critical issue in bulk transfer as it is in voice or data traffic. Data and image traffic tend to be bursty. Voice traffic is more regular. Video data require low delay and high throughput.

The performance characteristics of the different types of traffic are listed in Table 2.1 [16]. Communication traffics are categorized into two classes. Class I contains traffic with real time

Table 2.1: Characteristics of Different Types of Network Traffic

	Class I	Class II	
Type of traffic	Digitized voice, video,facsimile, sensor bulk data	Narrative/record, interactive, query/response, data base update	Nosensor bulk data
Call duration	Several minutes	Seconds to minutes	Minutes to hours
Error control	Generally none (possibly forward error correction for video and facimile)	Automatic repeat request, forward error correction/ automatic repeat request	May or may not be required
Cross network delay	Less than 200 msec	Less than 1 sec	Minutes to hours
Message length	$10^5 - 10^7$ bits	600–6000 bits	$10^6 - 10^8$ bits
Transmission rates	2.4–200 kbit/sec	45 BPS–100 kbit/sec	4.8–100 kbit/sec
Availability	Blocking	No blocking, but may be delayed	No blocking, but may be throttled

traffic which requires continuous real-time delivery, such as voice or video. Class II traffic is characterized by short, discrete data messages to long messages of store-and-forward type. This class of traffic usually can tolerate some delay.

2.1.2 Circuit Switching

Circuit switching in a communication network means there is a physical link ("copper path") between two communicating stations. This path is dedicated to the connection of these stations for the duration of communication. Of course, the physical link between two stations may be microwave links onto which thousands of paths are multiplexed. The property of the circuit switching is that once the connection is setup, a dedicated path exists until the communication is completed. Hence there is a need to set up an end-to-end path before any data is transmitted. Three phases are involved in the circuit switching.

1. *Circuit establishment:* the originating station sends out a request to establish a link to the destination station. If, for any reason, the link cannot be established, a busy signal is returned to the originating station.
2. *Data transfer:* When the link is established, the stations start to transmit data.
3. *Circuit disconnect:* Connection is terminated after a period of time. This is originated by one of the two stations.

A delay is expected in the first phase of circuit establishment because the connecting request needs to propagate to the destination, and be acknowledged. However, there is no delay other than propagation delay once a connection is setup. The network is effectively transparent to the users in the second phase. In effect, the network provides a “pipeline” for the two stations. However, there are two drawbacks [28]:

1. In a typical terminal-to-host data connection, much of the time the line is idle. Thus, with data connections, a circuit-switched approach is inefficient.
2. In a circuit-switched network, the connection provides for transmission at a set of fixed data rate. This limits the flexibility of the network in transmission of the variety of different bandwidths.

2.1.3 Packet Switching

Another type of switching is *packet switching*, in which a block of data is broken into small, possibly-fixed size cells called packets. Each packet contains a portion of user’s data and control information. There is no dedicated path and guaranteed bandwidth established in advance between sender and receiver. Instead, at each node *en route*, packets are received, stored briefly in the buffer, and passed on to the next node. There are two approaches used in packet switching networks:

1. *datagram approach*: Each packet is treated independently. There is no connection between packets, even they are all from the same source and going to the same destination. Thus packets from the same source can take different routes to reach the same destination. A packet sent early can arrive late just because it took a longer route. It is also hard to detect a lost packet since the destination node does not know the routing of packets.
2. *virtual circuit approach*: A route is established before any packet is sent. Packets from one source node to same destination node follow the same route. This guarantees the order of packets and provides some error control mechanism.

Over all, packet switching has the several advantages over circuit switching.

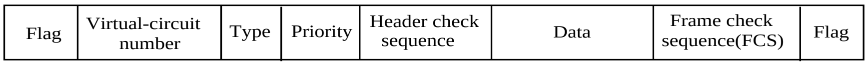
1. It is a more efficient way to use the line for data.
2. It provides flexible speed conversion. Two stations of different data rates can exchange packets. The network buffers the data and delivers it at the appropriate data rate.
3. A host can converse with a number of terminals over a single line simultaneously.

However, packet switching has some disadvantages compared to circuit switching.

1. Complex routing and congestion control.
2. The delay time varies, and is a function of load.

2.2 Fast Packet Switching

Due to demands for higher and faster networks, a switching technique capable of handling high-speed transmission with low error rates has been developed. This concept, *fast packet switching*, is an adaptation of packet switching for use in high-speed environments. The traditional packet switching technique is adopted because it is independent of data rate and accommodates bursty traffic. To address the drawbacks of the packet switching such as variable delay and throughput



Flag	Virtual-circuit number	Type	Priority	Header check sequence	Data	Frame check sequence(FCS)	Flag

Figure 2.1: Fast Packet Switching Format

bottlenecks, several additional features have been incorporated into the fast packet switching technique.

1. No link-by-link error control.
2. No link-by-link flow control.
3. End-to-end error control if necessary.
4. Use of internal virtual circuit.
5. Hardware switching.

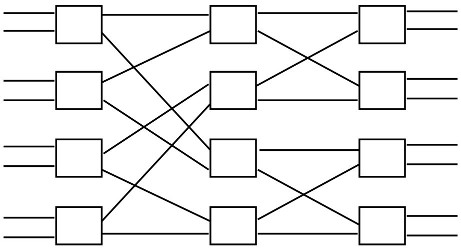
The traditional error control at the data-link level is no longer needed due to the high quality and speed of the modern digital transmission trunks. The use of flags and a Frame Check Sequence (FCS) is sufficient for error detection. If an error is detected, the packet is simply discarded. There is no hop-level retransmission. Error control can be implemented at higher level protocol. Thus fast packet switching provides only end-to-end error recovery mechanism. A virtual-circuit number field is still required to provide routing information of the virtual circuit. An example of the fast packet switching frame is shown in Fig 2.1 [19, 29]. In fast packet switching, the basic switching function is implemented by hardware. The hardware configuration that best fulfills this purpose is a multi-stage interconnection network.

2.3 Multi-stage Interconnection Networks

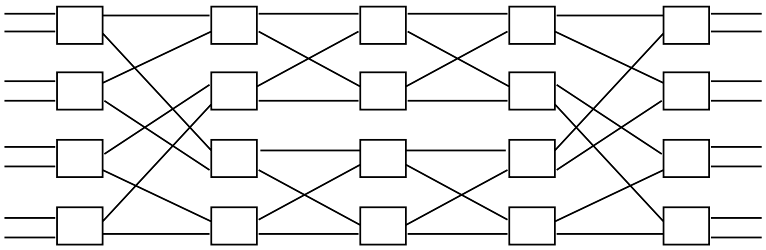
Multi-stage interconnection networks (MINs) have been studied in multiprocessor applications and communication systems. In particular, Feng [5] in his survey discussed the various topologies and communication protocols of the MINs. A multistage network consists of more than one stage of switching elements and is usually capable of connecting an arbitrary input terminal to an arbitrary output terminal. Among MINs, those with blocking property are most common ones. Figure 2.2 shows some examples of MINs.

A banyan network is one type of interconnection networks. It usually consists of 2-input and 2-output switching elements connected together in stages. An $N \times N$ banyan network is composed of $log_2 N$ stages with $N/2$ switching elements in each stage. There is only one path from any input to any output port. The routing of a packet is done by individual switching element within the switch fabric. Each packet has an $n-$bit header in an $n-$stage switch. The switching element at stage 1 (shown as a small box in the figure) routes the packet up or down according to the first bit of the header ("zero" or "one" indicate up or down respectively). It then removes the first bit from the header. The succeeding switching elements perform the same routing function based on the information in the packet header until the packet arrives at its destination port. At each stage, different packets may be treated differently depending on the state of the switching element and the congestion condition at the later stage. This characteristic of routing by switching element is called "self-routing". The self-routing property enable the switch to operate in either synchronous or asynchronous mode. Indeed, the control of the network is distributed. The merit of this configuration is that the network can be constructed modularly and can be easily implemented by using VLSI technology, reducing the cost of the switching fabric. Since several packets can be switched at the same time in parallel, the switch is able to handle a higher traffic load. Nevertheless, some problems need to be addressed. Most important of all, the problem in which packet blocking caused by contention of the same internal link, by destined to the same output port or simply by waiting in the input queue where the head of the queue is blocked due to one of the two previous reasons. This blocking can generate a local congestion point and reduce the performance of the switch.

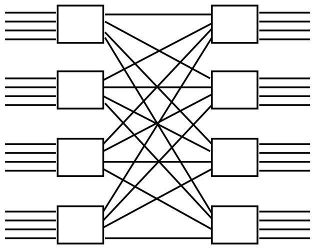
To minimize the problem of packet blocking, a pre-processing network can be used. The pre-processing network can be either a sorting network, as in Huang and Knauer [8] and Hui and



8x8 Baseline Network



8x8 Benes Network



16x16 Delta Network

Figure 2.2: Examples of Multi-stage Interconnection Network

Arthurs [9], or distribution network, as in Turner [30] and Newman [22, 23]. In particular, Hui and Arthurs [9] used a Batcher network in front of the banyan network to first sort the connection requests. After sorting, the conflicting requests are adjacent to each other, and a request wins the contention if the request above it in the sorted order is not for the same output port. The winning requests then send back acknowledgments to their inputs. Input ports, upon receiving the acknowledgments, transmit the full packets through the network. Input ports which fail to receive an acknowledgment retain the packet in a buffer for retry in the next time slot. Huang and Knauer [8] suggested a similar approach, except that the whole packets instead of connection requests were sent through the network. The packets which lose the contention are concentrated by a concentration network and fed back to the front end of the Batcher sorting network for reentry. Turner [30] in his "integrated services packet network" proposed a distribution network in front of the switching network. The function of distribution network is to distribute packets evenly across all its output ports. This is done by having each switch element route packets alternately out its two ports. In his paper, Turner used the same banyan network as a distribution network. In contrast, Newman [23] used a Benes network as his distribution fabric in front of a modified banyan network which served as a routing fabric.

2.4 Contention Resolution Algorithms

When blocking occurs, an algorithm is needed to determine what happens to the blocked packet. Two approaches are usually taken. The first approach is to hold the packet at the previous switching element, provided there is enough buffer space to save the packet. The second approach is that we simply discard the blocked packet to make way for the following packets. The discard packet needs to be retransmitted. This is done usually by upper layer protocol.

The first approach, which we call *back-pressure algorithm* retains blocked packets in the buffers at the previous stage of the switching fabric. When a buffer at a particular stage is full, the packets at the previous stage destined to this switching element are delayed. The internal buffers between two stages are usually small. Single space buffers are quite common [10, 32]. Therefore, it is possible that the blocking in one switching element causes a chain effect which propagates to input queues of the switch fabric. In other words, this chain effect is caused by the back-pressure from the blocked packet in the switching element's buffer. The size of the buffer in the switching element affects the performance of the switch. This has been studied analytically and using simulation by several scholars [2, 3, 4, 10, 11, 14, 24, 30, 32].

In the second approach, which we call *reject algorithm*, the blocked packet is dropped whenever there is a contention for the link. For some types of packet, such as voice, retransmission is not necessary. For other types of packet, the retransmission is needed and is done by the upper layer protocol. However, the retransmission produces the problem that packets may arrive out of order and must be reordered by the receiving host.

CHAPTER 3

STOCHASTIC ACTIVITY NETWORKS

3.1 Model Definition and Execution

Stochastic activity networks (SANs)[18, 21] are extensions of activity networks(ANs). They incorporate features of both stochastic Petri nets [20] and queueing models. SANs were developed to facilitate unified performance/dependability evaluation. It also includes the features which permit the representation of parallelism, timeliness, fault tolerance, and degradable performance [17].

SAN structures are made up of four types of primitives:*places, activities, input gates,* and *output gates*(see Figure 3.1. *Places* are denoted by circles. *Activities* ("transitions" in Petri net terminology) can be either *timed* or *instantaneous.* Timed activities represent activities in the modeled system whose durations affect the system's ability to perform. Instantaneous activities represent those activities of the system which complete in a small amount of time, relative to the performance variables considered Timed and instantaneous activities are depicted as elongated ovals and vertical bars respectively. There are *cases* associated with activities to realize uncertainty as to what happens when an activity completes. Cases are represented by hollow dots attached to one side of the ovals or bars. Each small circle depicts one case. Gates, denoted as triangles, provide greater flexibility in describing rules of enabling and completing activities. Input gates consist of a finite set of input places and are connected to a single activity. Output gates are connected to a single activity and a finite set of output places. Each input gate has associated

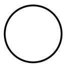
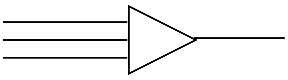
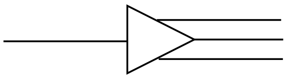
	SAN SYMBOL
Place	
Input Gate	
Output Gate	
Timed Activity	
Instantaneous Activity	

Figure 3.1: Symbols for Stochastic Activity Network

with it an *enabling predicate* and an *input function*. The enabling predicate specifies the enabling rules for the associated activity. Each output gate contains an *output function*. The input and output functions describe the changes which result from the completion of the associated activity. The state of a SAN is defined by its “marking”. A marking is an assignment of “tokens” to the places of a SAN.

A stochastic activity network is an interconnection of finite number of these primitives, subject to the following connection rules [21]:

1. Each input of an input gate is connected to a unique place and the output of an input gate is connected to a single activity.

2. Different input gates of an activity are connected to different places.
3. Each output of an output gate is connected to a unique place and the input of an output gate is connected to a single activity (via some case).
4. Different output gates of an activity for a case are connected to different places.
5. Each place and activity is connected to some input gate or output gate.

The stochastic nature of a SAN is realized by associating an *activity time distribution function* with each timed activity and a *probability distribution* with each set of cases. Generally, both distributions can depend on the global marking of the network. A mechanism for restarting activities that have been activated is also provided in the nets. A *reactivation function* [18] specifying a set of *reactivation markings* for each marking is included in each timed activity. Given that an activity is activated in a specific marking, the activity is *reactivated* every time a marking in the set of reactivation markings is reached.

SANs execute in time through completions of activities which result in changes in the markings. Specifically, an activity is chosen to *complete* in the present marking based on the relative priority among activities (instantaneous activities take presidence of timed activities) and the activity time distributions of *enabled* activities. A case of the activity is then selected and completed based on the case distribution the activity. This uniquely determines the next marking of the network. The marking is obtained by first executing the function in each input gate (input function) connected to the input of chosen activity and then executing the function in each output gate (output function) of the chosen case. Once the new marking is obtained, the procedure repeats itself. A detailed discussion for the SAN execution can be found in [26].

Stochastic activity networks can be solved by both analysis and simulation, depending on system characteristics. Informally, SANs can be solved via analytic methods when all activity time distributions are exponential and activities are reactivated often enough to ensure that their rates depend only on the current state. When this is the case, the analytic solutions for a wide class of behavior variables can be obtained through solution of appropriate stochastic processes. Simulation can alos be used to solve for SAN behavior.

The complexity of the evaluation procedures and the sizes of the base models requires their implementation in software. A software package, called METASAN[1] , has been developed to address this need.

3.2 Evaluation Tool – METASAN

METASAN [27] was written using UNIX tools (C, Yacc, Lex, and Csh) and consists of about 37,000 ines of source code. Solution options include analytical techniques (applicable under certain well defined conditions) as well as both terminating and steady– state simulation.

A user to interacts with METASAN through a menu. Two files are needed to execute a METASAN model: a structure file and an experiment file. The structure file is a direct translation of the SAN into a textual form that can be understood by the package. The experiment file specifies the solution algorithm and the performance variables. These files are accessible from the menu. Model construction consists of describing the structure of the system to be modeled using the editor, compiling the description, describing the experiment file, and compiling the experiment file. The result of these actions is a machine understandable description of the system to be modeled and the desired performance variables. Then user selects the desired solution options and executes the model.

The SAN description language, *Sanscript*[2] , permits a SAN to be specified in a textual form understandable to the (SAN) complier. Sanscript also permits easy specification of complex enabling predicates, activity time functions, reactivation functions, and gate functions. At a high–level, a Sanscript description consists of four parts: a header, local variable declarations, definition

---

[1] METASAN is a Trademark of the Industrial Technology Institute.

[2] Sanscript is a Trademark of the Industrial Technology Institute

of all the primitives used, and a specification of all function values and interconnection associated with each primitive. A host of activity time distribution types are available, representing all service distributions normally used in evaluation. Complex activity time distribution parameters, case distributions, gate predicates and functions, and reactivation functions may be specified using the *"MARK"* function and a few lines of C code. Here the notation *"MARK(place)"* refers to the current marking of place *"place"*. Many stochastic activity networks contain numerous similar subnetworks that are replicated many times. Construction of these subnetworks directly in Sanscript would be tedious. To simplify such construction a macro-preprocessor for METASAN is provided. This preprocessor allows one to define subnetworks once in a parameterized manner, an then construct a specific subnetwork via a single macro call. After the specification of the SAN in Sanscript is complete, it is passed to the SAN compiler and translated into an internal form understandable by the solution modules.

The experiment file has great flexibility in the definition of performance variables. Unlike many modeling packages which limit evaluations to a few pre-defined variables (e.g. queue length, server utilization), METASAN permits the specification of complex user–defined performance variables. Performance variables specified for solution by simulation are based on the notion of a *path*. A path is a sequence of marking–activity–case triples which define a possible behavior on the net. Events such as initiations of paths, completions of paths, and traversals of paths are then naturally defined. Definition of these events make it possible to estimate a variety of time related characteristics of path sets. All convertional performance variables plus a wide class of unconvertional variables can be represented in this framework.

A variety of analytic solvers are implemented in the package. Steady–state state occupancy probabilities are obtained either by Gaussian elimination or by Gauss-Seidel iteration, depending on the size of the state–space and convergence characteristics of the particular model. Reward model solution techniques are also implemented.

Solution via simulation is also supported by METASAN. Simulation is normally used when solution of the base model via analytic means becomes intractable. This can occur, for example, when complex reactivation functions are specified, activity time distributions are general, the desired performance variables are sufficiently complex, or the state space is extremely large. Both the terminating and steady–state simulation solvers are based on a discrete–event next–event time advance simulator core. Currently, two methods for confidence interval estimation are supported. The first is an iterative method based on the replication approach, and is used or terminating simulations. Using this method, one specified the relative precision and level of confidence desired as part of the experiment file input. The second method is used for steady–state simulations and is an iterative batching procedure, where the user must specify the length of initial transient, bathc size, relative precision desired, and level of confidence desired. This simulation package was chosen as the evaluation tool for this research.

CHAPTER 4

MODELING SWITCHES

4.1 Switch Models

Three designs of Multistage Interconnection Network are presented here: a basic 8×8 banyan network (baseline network), 8×8 modified delta network, and 8×8 modified network with 4×4 banyan network, multiplexers and demultiplexers(Mux-Demux). These designs are represented by the stochastic activity network first and then translated into descriptions understandable to METASAN. METASAN is then used to obtain the results which illustrate the performance of different designs. Due to the complexity of the models, the simulation solver in METASAN is used.

4.1.1 General Model Assumptions

As mentioned in previous sections, the switching elements in banyan network can have no buffer, single, or multiple buffers. Furthermore, the input queue is usually short. Therefore, our models assume each input queue holds 3 packets. Each input queue obeys first in first out queueing discipline. Single internal buffer is placed before each switching element. Packet arrival rate λ is assumed to have a Poisson behavior. Packet processing time in each switching element is assumed to be the normally distributed with very little variance (1/100).

It is assumed that each input controller operates asynchronously in packet level. The minimum delay is achieved when a packet proceeds to next stage without waiting anywhere in the entire network. Thus the minimum delay is $n \times (\textit{processing time})$ where n is the number of stages in the network.

In the back-pressure algorithm, the control signals are assumed to be passed across the network from the current stage to the previous stage in the switch so that input port at different stages is able to determine whether to send or hold the current packets. The packets are lost at the input queues only when the queues are full. Otherwise, the packets are sent according to the destination address within the packets.

In the reject-retransmission algorithm, packets are routed according to the destination address. However, there is no control signal being passed back from subsequent stage. The switching elements forward the packets at all circumstances. If the input buffer is full at the certain element, that packet is simply rejected and dropped. This packet then is re-transmitted again using a possibly different route.

We now examine each of the three interconnection networks designs in more detail.

4.1.2 Model of Basic 8×8 Banyan Switching Network

An 8×8 banyan network, containing elements of 2×2 cross–bar switch, is shown in Figure 4.1. This is a three–stage network which contains 4 switch elements at each stage. Each stage is connected to the adjacent stage in the fashion that any of the input ports can be connected to any of the output ports. Since network operates simultaneously, there is no holding back for the packets coming to different input ports except the packets going to the same output ports.

Packets are assumed to arrive at an input buffer controller at a rate of λ. If the corresponding buffer of the switching element at first stage is able to accept a packet, the input buffer controller will allow a packet into the input buffer. Otherwise, the packet is rejected. The input buffer can hold only three packets at one time. The packets are placed in the input buffers in the order of their arrivals, and they are passed to the first stage of banyan network in first-in-first-out mode.

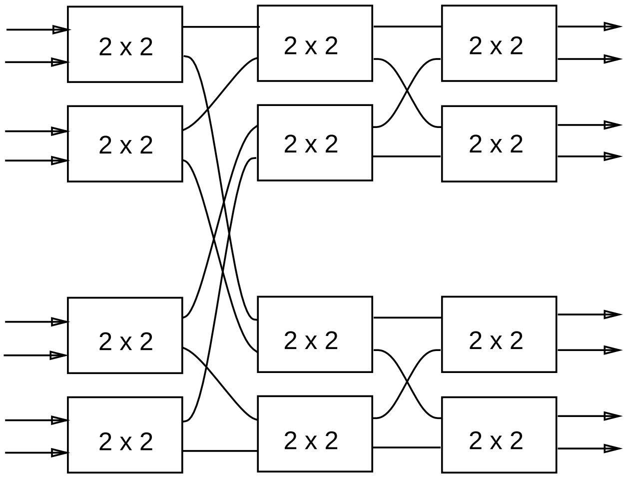


Figure 4.1: 8 × 8 Basic Banyan (Baseline) Network

Each switching element in the first stage routes the packet according to the first bit of the address. "Zero" in the first bit will cause the packet to be routed to the upper output port of the switching element, while "one" will cause the packet to be routed to the lower output port. The packet is stored temperately in the buffer of the second stage switching elements, which only holds one packet. The second stage switching element will perform the routing according to the second bit of the destination address and then save the packets in the buffers of the third stage switching elements. The switching elements in the third stage do the same routing function using the third bit. The packets are routed to the outputs of the third stage elements, which are the destinations. Thus so ends the journey of these packets.

The SAN representation of a 2 × 2 switching element is shown in Figure 4.2. It consists of four places, two timed activities and two associated input gates. The *input_1* and *input_2* places represent the single input-buffers of the 2 × 2 cross-bar switching element. Gates *ck_1* and *ck_2* check the content of the buffers. If there is a packet in the input buffer (token in *input_1* or *input_2*) and the designated output buffer for that packet (*output_1* or *output_2*) is available (empty), the timed activity (*processing_1* or *processing_2*) is enabled. Upon completion of the activity, the packet (tokens) in the input buffer is moved to the proper output buffer which in turn is the

Table 4.1: Activity Parameters for 2 × 2 Switching Element

Activity	**Rate**	**Probability**
processing_1	normal(μ_1)(δ_1)	1
processing_2	normal(μ_2)(δ_2)	1

Table 4.2: Input Gate Parameters for 2 × 2 Switching Element

Gate	**Enabling Predicate**	**Function**

ck_1	(MARK(*input_1*)== 1 and MARK(*output_1*)==0) or
	(MARK(*input_1*)== 2 and MARK(*output_2*)==0)
		if (MARK(*input_1*)==1)
		MARK(*output_1*)=1;
		MARK(*input_1*)=0;
		else
		MARK(*output_2*)=2;
		MARK(*input_1*)=0;

ck_2	(MARK(*input_2*)== 1 and MARK(*output_1*)==0) or
	(MARK(*input_2*)== 2 and MARK(*output_2*)==0)
		if (MARK(*input_2*)==1)
		MARK(*output_1*)=1;
		MARK(*input_2*)=0;
		else
		MARK(*output_2*)=2;
		MARK(*input_2*)=0;

input buffer of the switching element at next stage. The address of packet is modeled by different numbers of tokens in places. In the case of 2 × 2 switch, there are only two types of packets, i.e., one that goes to the upper output port and one that goes to the lower. They are modeled by single token and two tokens in a place. *processing_1* and *processing_2* represent the processing time needed for the switching element. Each of them has a normally distributed activity time with mean μ and variance δ. The mean μ varies to reflect different rates of time needed for processing one packet, while variance is always set to one-hundredth of mean ($\delta = \frac{1}{100}\ \mu$). The distributions of the activities, gate functions and predicates for the this SAN model are listed in Table 4.1 and Table 4.2.

Figure 4.3 presents the SAN model for a 8×8 Baseline Banyan network. A detailed diagram of part A of Figure 4.3 is shown in Figure 4.4. A detailed diagram of part B is shown in Figure 4.5.

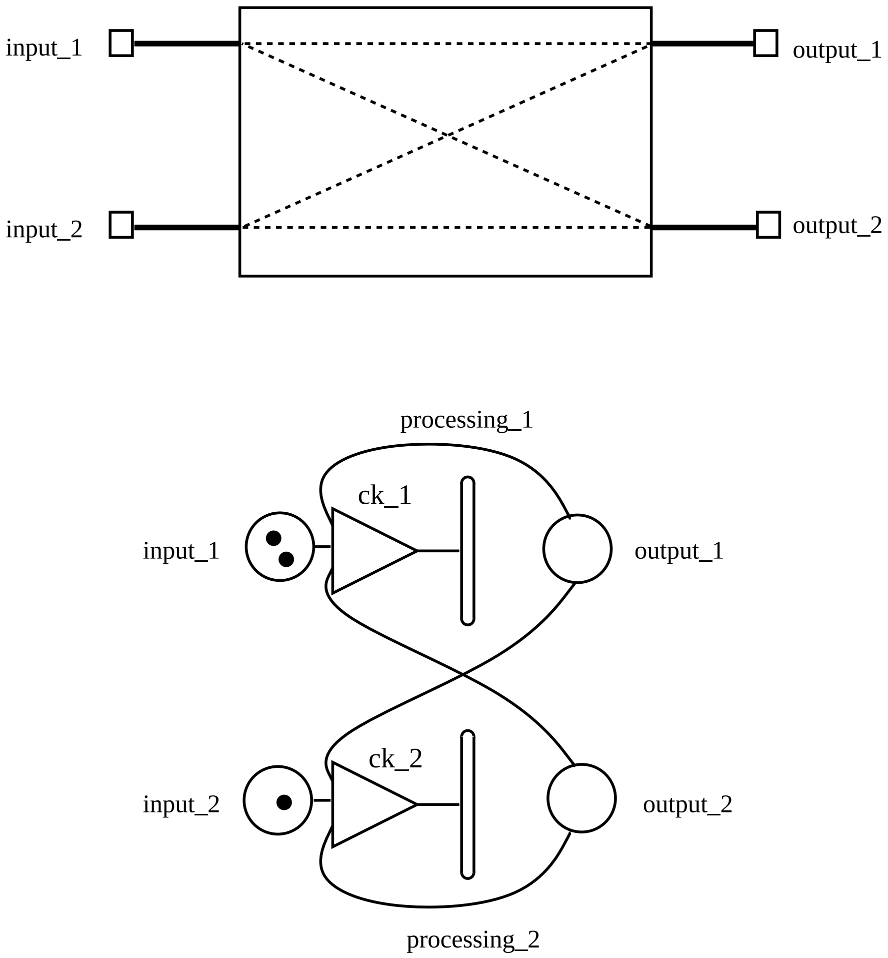


Figure 4.2: SAN representation for 2 × 2 switching element

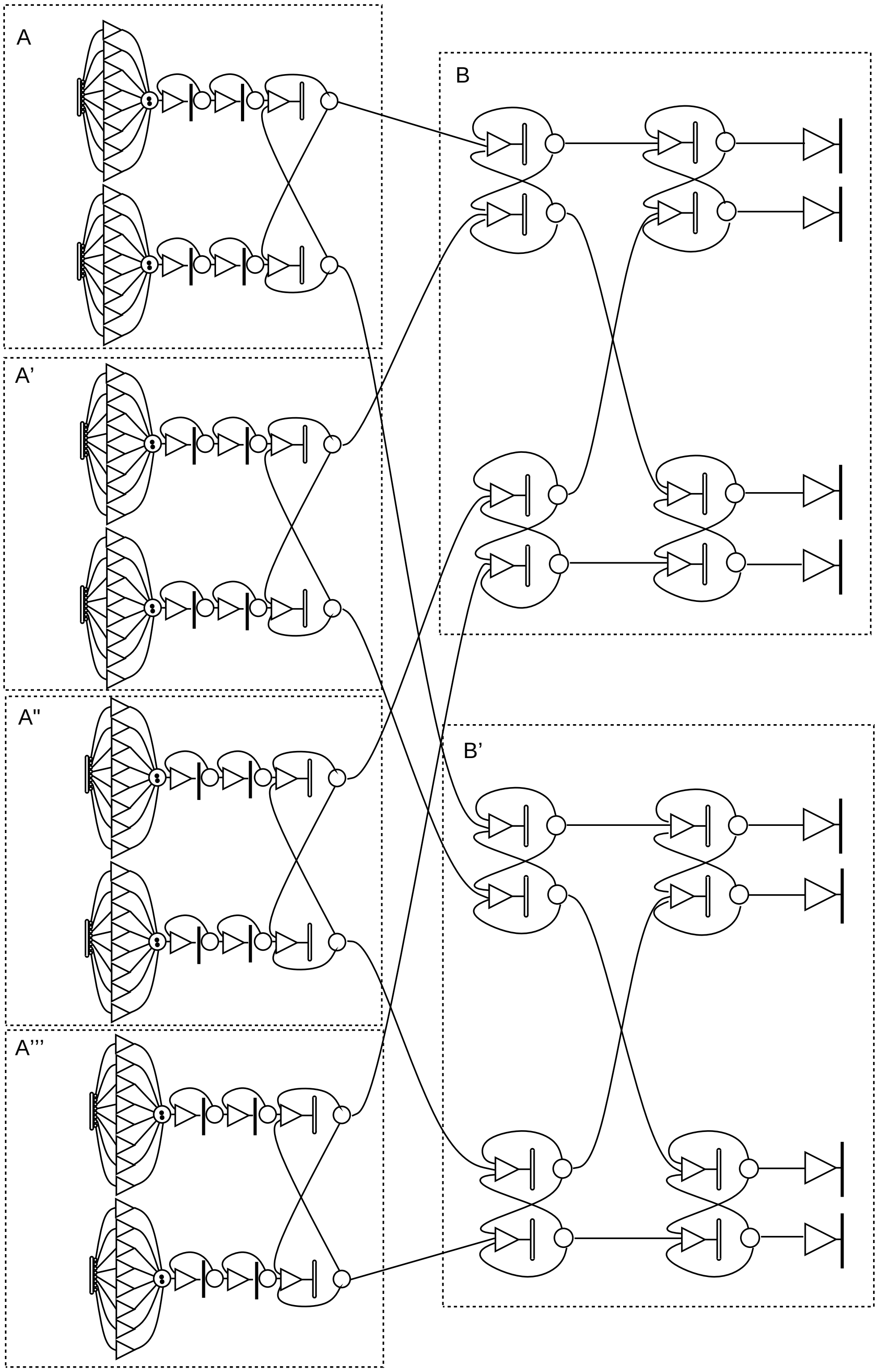


Figure 4.3: SAN Representation for 8 × 8 Baseline Switch

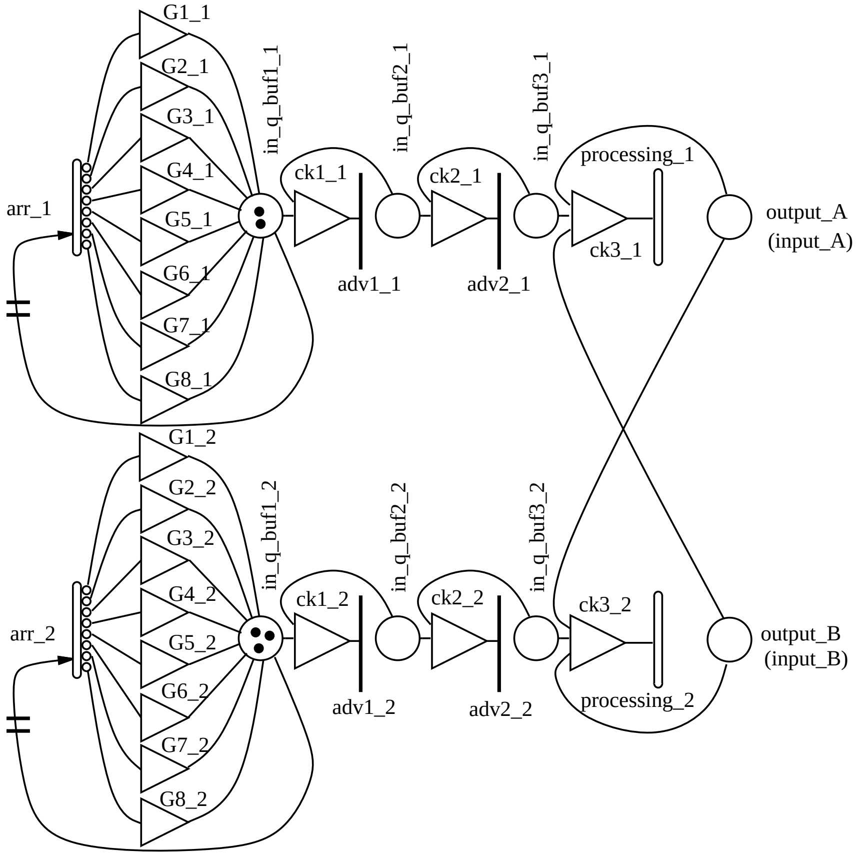


Figure 4.4: Detailed Diagram of Part A in 8 × 8 Baseline Switch

Table 4.3: Output Gate Parameters for Part A of 8 × 8 Baseline Switch SAN

Gate	**Function**
$G1\_i$	MARK($in\_q\_buf1\_i$) = 1;
$G2\_i$	MARK($in\_q\_buf1\_i$) = 2;
$G3\_i$	MARK($in\_q\_buf1\_i$) = 3;
$G4\_i$	MARK($in\_q\_buf1\_i$) = 4;
$G5\_i$	MARK($in\_q\_buf1\_i$) = 5;
$G6\_i$	MARK($in\_q\_buf1\_i$) = 6;
$G7\_i$	MARK($in\_q\_buf1\_i$) = 7;
$G8\_i$	MARK($in\_q\_buf1\_i$) = 8;

The timed activities *arr*'s represent the arrivals of the packets to the system and according to the assumption made above, have an exponential activity time distribution with rate α. In our evaluations, the rate of arrival is always assumed to be one ($\alpha = 1$). Eight cases connecting to output gates *G1*'s,*G2*'s,. . .,*G8*'s are associated with each timed activity *arr*. The output gate function, when executed, generates tokens which represent the address of packet destination. One token in a place stands for a packet which is designated to output port 1, represented by place *output_1*. Two token stands for a packet addressed to output port 2, *output_2*, and so on. Places *in_q_buf1*'s, *in_q_buf2*'s and *in_q_buf3*'s are three buffer spaces in the input queues. Instantaneous activities *adv1*'s and *adv2*'s along with input gates *ck1*'s and *ck2*'s are the SAN representations of FIFO queueing discipline to account for different types of packets. Presence of one or more token in place *in_q_buf1* and vacancy of the next place *in_q_buf2* enables the instantaneous activity *adv1*. *adv1* completes immediately and the input function of gate *ck1* moves all tokens in *in_q_buf1* to *in_q_buf2*. Input gate *ck2* examines the content of places *in_q_buf2* and *in_q_buf3* to enable/disable the instantaneous activity *adv2*. Upon completion of *adv2*, *in_q_buf2* is vacated and tokens are moved to *in_q_buf3*. The output gate functions for this SAN are listed in Table 4.3. The input gate predicates and functions for this SAN can be found in Table 4.4. The activity parameters are shown in Table 4.5. Part A', A" and A"' are exactly the same as part A.

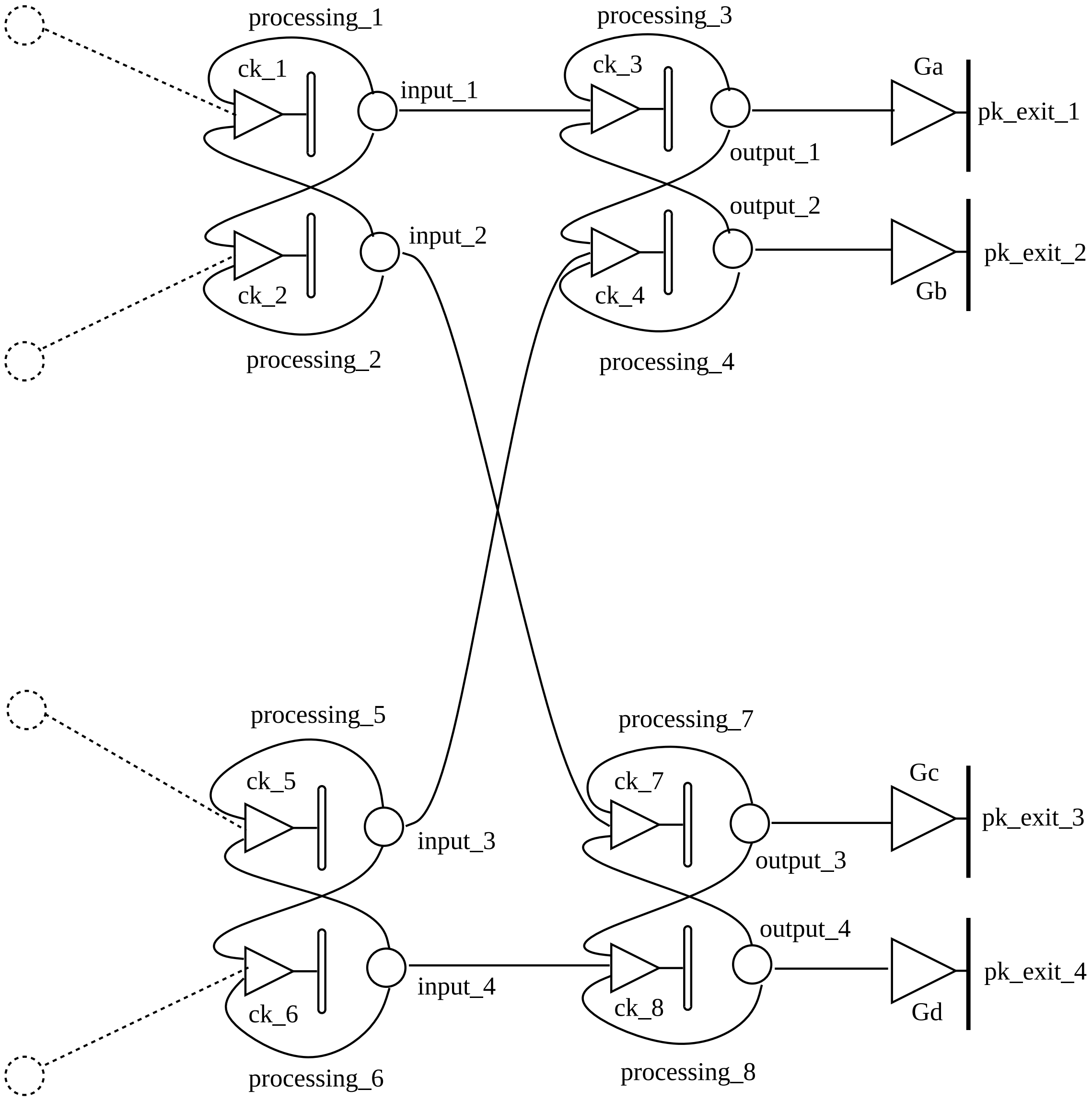


Figure 4.5: Detailed Diagram of Part B in 8 × 8 Baseline Switch

Table 4.4: Input Gate Parameters for Part A of 8 × 8 Baseline Switch SAN

Gate	**Enabling Predicate**	**Function**
$ck1\_1$	MARK($in\_q\_buf1\_1$) > 0 and MARK($in\_q\_buf2\_1$) $== 0$	MARK($in\_q\_buf2\_1$)= MARK($in\_q\_buf1\_1$); MARK($in\_q\_buf1\_1$)=0;
$ck1\_2$	MARK($in\_q\_buf1\_2$) > 0 and MARK($in\_q\_buf2\_2$) $== 0$	MARK($in\_q\_buf2\_2$)= MARK($in\_q\_buf1\_2$); MARK($in\_q\_buf1\_2$)=0;
$ck2\_1$	MARK($in\_q\_buf2\_1$) > 0 and MARK($in\_q\_buf3\_1$) $== 0$	MARK($in\_q\_buf3\_1$)= MARK($in\_q\_buf2\_1$); MARK($in\_q\_buf2\_1$)=0;
$ck2\_2$	MARK($in\_q\_buf2\_2$) > 0 and MARK($in\_q\_buf3\_2$) $== 0$	MARK($in\_q\_buf3\_2$)= MARK($in\_q\_buf2\_2$); MARK($in\_q\_buf2\_2$)=0;

$ck3\_1$	($1 \leq$ MARK($in\_q\_buf3\_1$) ≤ 4 and
	MARK($output\_A$)==0) or
	($5 \leq$ MARK($in\_q\_buf3\_1$) ≤ 8 and
	MARK($output\_B$) == 0)
		if ($1 \leq$ MARK($in\_q\_buf3\_1$) ≤ 4)
		MARK($output\_A$)= MARK($in\_q\_buf3\_1$);
		MARK($in\_q\_buf3\_1$) = 0;
		else
		MARK($output\_B$) = MARK($in\_q\_buf3\_1$);
		MARK($in\_q\_buf3\_1$) = 0;

$ck3\_2$	($1 \leq$ MARK($in\_q\_buf3\_2$) ≤ 4 and
	MARK($output\_A$)==0) or
	($5 \leq$ MARK($in\_q\_buf3\_2$) ≤ 8 and
	MARK($output\_B$) == 0)
		if ($1 \leq$ MARK($in\_q\_buf3\_2$) ≤ 4)
		MARK($output\_A$)= MARK($in\_q\_buf3\_2$);
		MARK($in\_q\_buf3\_2$) = 0;
		else
		MARK($output\_B$) = MARK($in\_q\_buf3\_2$);
		MARK($in\_q\_buf3\_2$) = 0;

Table 4.5: Activity Parameters for Part A of 8 × 8 Baseline Network

Activity	**Rate**	**Probability**							
		case							
		1	2	3	4	5	6	7	8
arr_1	exp(λ)	0.125	0.125	0.125	0.125	0.125	0.125	0.125	0.125
arr_2	exp(λ)	0.125	0.125	0.125	0.125	0.125	0.125	0.125	0.125
$adv1\_1$	inst	1	-	-	-	-	-	-	-
$adv2\_1$	inst	1	-	-	-	-	-	-	-
$adv1\_2$	inst	1	-	-	-	-	-	-	-
$adv2\_2$	inst	1	-	-	-	-	-	-	-
$processing\_1$	normal(μ)(δ)	1	-	-	-	-	-	-	-
$processing\_2$	normal(μ)(δ)	1	-	-	-	-	-	-	-

Figure 4.5 shows a stochastic activity network of 4 × 4 unit in 8 × 8 switch model. This is a detailed drawing of part B in Figure 4.3. Each switching element in the unit has the same processing time, represented by the activity time of each activity *processing*. The enabling predicates of the input gates are the same–when an input buffer is filled and output buffer is free, the gate holds. The gate functions of input gates *ck*'s move the tokens according to the packet address (number of tokens). The input gate predicates, functions and activity parameters can be found in Tables 4.6, 4.7 and 4.8, respectively. Input gates G_a, G_b, G_c and G_d check whether the places *output*'s are occupied. If so, instantaneous activities *pk_exit*'s are enabled. When these activities complete, the gate functions in G_a, G_b, G_c and G_d remove all the token in related *output*'s and subtract one token each from place *packet_in_system* (not shown in the figures) which represents packets exiting the switch. The place, *packet_in_system*, is an internal counter which holds the number of packets in the system at any moment. Part B' in Figure 4.3 is the same as part B except the input gates *ck_1*,..., *ck_6* now check for the packet addresses from 5 to 8 instead of from 1 to 4.

4.1.3 Model of 8 × 8 Modified Delta Network

Delta networks [24, 25], also known as shuffle-exchange networks [14], are the second class of banyan network studied. Generally, a delta network of size N consists of $log_d N$, with N/d switching elements per stage. A typical 16×16 delta network can be found in Figure 2.2. Newman [23] proposed a modified version of delta network in which interconnection links are replicated in order to build networks with size that is an integer power of 2. A 16 × 16 modified delta network of switching elements of degree 8 is shown in Figure 4.6. The modified delta network, under the rigid definition, does not belong to the class of banyan networks since there is more than one path existing between any pair of input and output.

Table 4.6: Input Gate Parameters for Part B of 8 × 8 Baseline Switch Model

Gate	**Enabling Predicate**	**Function**

ck_1	($1 \le$ MARK(*input_A*) ≤ 2 and
	MARK(*input_1*)==0) or
	($3 \le$ MARK(*input_A*) ≤ 4 and
	MARK(*input_2*) == 0)
		if ($1 \le$ MARK(*input_A*) ≤ 2)
		MARK(*input_1*)= MARK(*input_A*);
		MARK(*input_A*) = 0;
		else
		MARK(*input_2*) = MARK(*input_A*);
		MARK(*input_A*) = 0;

ck_2	($1 \le$ MARK(*input_A′*) ≤ 2 and
	MARK(*input_1*)==0) or
	($3 \le$ MARK(*input_A′*) ≤ 4 and
	MARK(*input_2*) == 0)
		if ($1 \le$ MARK(*input_A′*) ≤ 2)
		MARK(*input_1*)= MARK(*input_A′*);
		MARK(*input_A′*) = 0;
		else
		MARK(*input_2*) = MARK(*input_A′*);
		MARK(*input_A′*) = 0;

ck_3	(MARK(*input_1*) == 1 and
	MARK(*output_1*)==0) or
	(MARK(*input_1*) == 2 and
	MARK(*output_2*) == 0)
		if (MARK(*input_1*) == 1)
		MARK(*output_1*)= MARK(*input_1*);
		MARK(*input_1*) = 0;
		else
		MARK(*output_2*) = MARK(*input_1*);
		MARK(*input_1*) = 0;

ck_4	(MARK(*input_3*) == 1 and
	MARK(*output_1*)==0) or
	(MARK(*input_3*) == 2 and
	MARK(*output_2*) == 0)
		if (MARK(*input_3*) == 1)
		MARK(*output_1*)= MARK(*input_3*);
		MARK(*input_3*) = 0;
		else
		MARK(*output_2*) = MARK(*input_3*);
		MARK(*input_3*) = 0;

ck_5	($1 \le$ MARK(*input_A*”) ≤ 2 and
	MARK(*input_3*)==0) or
	($3 \le$ MARK(*input_A*”) ≤ 4 and
	MARK(*input_4*) == 0)
		if ($1 \le$ MARK(*input_A*”) ≤ 2)
		MARK(*input_3*)= MARK(*input_A*”);
		MARK(*input_A*”) = 0;
		else
		MARK(*input_4*) = MARK(*input_A*”);
		MARK(*input_A*”) = 0;

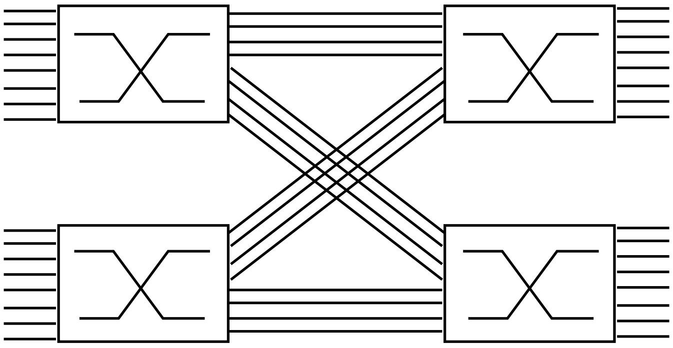


Figure 4.6: 16x16 Modified Delta Network

Table 4.7: Input Gate Parameters for Part B of 8 × 8 Baseline Switch Model (cont.)

Gate	**Enabling Predicate**	**Function**

ck_6	($1 \leq$ MARK($input\_A'''$) ≤ 2 and
	MARK($input\_3$)==0) or
	($3 \leq$ MARK($input\_A'''$) ≤ 4 and
	MARK($input\_4$) == 0)
		if ($1 \leq$ MARK($input\_A'''$) ≤ 2)
		MARK($input\_3$)= MARK($input\_A'''$);
		MARK($input\_A'''$) = 0;
		else
		MARK($input\_4$) = MARK($input\_A'''$);
		MARK($input\_A'''$) = 0;

ck_7	(MARK($input\_2$) == 3 and
	MARK($output\_3$)==0) or
	(MARK($input\_2$) == 4 and
	MARK($output\_4$) == 0)
		if (MARK($input\_2$) == 3)
		MARK($output\_3$)= MARK($input\_2$);
		MARK($input\_2$) = 0;
		else
		MARK($output\_4$) = MARK($input\_2$);
		MARK($input\_2$) = 0;

ck_8	(MARK($input\_4$) == 3 and
	MARK($output\_3$)==0) or
	(MARK($input\_4$) == 4 and
	MARK($output\_4$) == 0)
		if (MARK($input\_4$) == 3)
		MARK($output\_3$)= MARK($input\_4$);
		MARK($input\_4$) = 0;
		else
		MARK($output\_4$) = MARK($input\_4$);
		MARK($input\_4$) = 0;

Ga	MARK($output\_1$) > 0 and	MARK($packet\_in\_system$)= MARK($packet\_in\_system$)− 1; MARK($output\_1$)=0)
Gb	MARK($output\_2$) > 0 and	MARK($packet\_in\_system$)= MARK($packet\_in\_system$)− 1; MARK($output\_2$)=0)
Gc	MARK($output\_3$) > 0 and	MARK($packet\_in\_system$)= MARK($packet\_in\_system$)− 1; MARK($output\_3$)=0)
Gd	MARK($output\_4$) > 0 and	MARK($packet\_in\_system$)= MARK($packet\_in\_system$)− 1; MARK($output\_4$)=0)

Table 4.8: Activity Parameters for Part B of Baseline Switch

Activity	**Rate**	**Probability**
processing_1	normal(μ)(δ)	1
processing_2	normal(μ)(δ)	1
processing_3	normal(μ)(δ)	1
processing_4	normal(μ)(δ)	1
processing_5	normal(μ)(δ)	1
processing_6	normal(μ)(δ)	1
processing_7	normal(μ)(δ)	1
processing_8	normal(μ)(δ)	1
pk_exit_1	exp(λ)	1
pk_exit_2	exp(λ)	1
pk_exit_3	exp(λ)	1
pk_exit_4	exp(λ)	1

The second switching fabric we will examine is based on the modified delta network. It is an 8×8 switching fabric constructed by four 4×4 switching networks. The 4×4 switching network made of 2×2 switching elements is adopted here instead of the 4×4 switching element. The reason for this change is that the 2×2 switching element is more commonly used than 4×4 element, and hence it would probably be more cost effective. Figure 4.7 shows the block diagram of this modified 8×8 delta network.

The arriving packets are queued in the input buffers. The input port controller launches a packet when it is at the front of the queue. Since there are multiple paths available for a packet to reach its destination port, a searching algorithm is employed to determine the route. The controller would attempt to transmit the packet through each available paths in sequence until it successfully finds one. If all the routes fail, the packet is either held in the input queue or rejected. After packets are launched into the switching fabric, the switching element, with single buffer, queues packet and send it to the next switching element. If the buffer of the next switching element is full, the packet is retained at the current switching element or simply discarded.

A SAN representation is shown in Figure 4.8. The ARR block of the switch model is shown

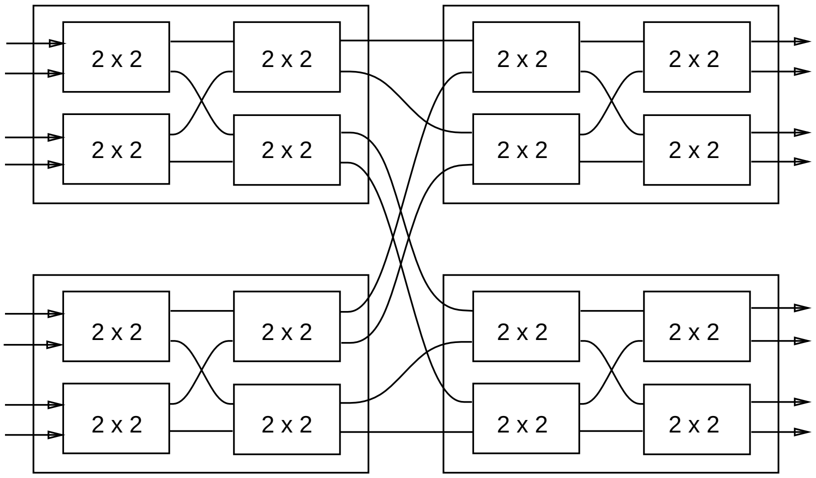


Figure 4.7: 8 × 8 Modified Delta Network

in Figure 4.9, which represents the SAN for the packet arrival, *arr*. Figure 4.10 depicts Part B in Figure 4.8. Part C in Figure 4.3 is the same as Part B in SAN for the 8 × 8 baseline switch and is shown in Figure 4.5. Packet arrival is modeled by the timed activity *arr* with eight cases. Each case represents the arrival of packet with different address. An output gate is connected to each case. When the case is chosen, the output gate completes and put certain number of tokens (the address label) in place *in_q_buf1*. There are total of eight output gates, *G1*,...,*G8*. When case 1 is chosen, *G1* generates 1 token in place *in_q_buf1_i*. Similarly, when case 8 is chosen, *G8* puts 8 tokens in the place. All ARR blocks in Figure 4.8 are the same. In part B, the place *in_q_buf1_i*, *in_q_buf2_i* and *in_q_buf3_i* represent the three stage input buffer(Figure 4.10). The instantaneous activities *adv1_i*, *adv2_i* and *adv3_i*, as mentioned in previous section, represent the advance of a packet from one buffer to another toward the front of queue. *ck1_i*'s and *ck2_i*'s, the input gates associated with these *adv*'s, check for the existence of token(s) in places connected to them. Each input gate is connected to two places $in\_q\_buf_i$ and $in\_q\_buf_{i+1}$. If place $in\_q\_buf_{i+1}$ is empty and $in\_q\_buf_i$ is occupied, the tokens are forwarded from $in\_q\_buf_i$ to $in\_q\_buf_{i+1}$ immediately. At the front of the queue, *in_q_buf3_i*, a packet is processed by a first stage 2 × 2 switching element. Input gates *ck3_1* *ck3_2*, *ck3_3* and *ck3_4* examine the number of tokens in places *in_q_buf3_1*,

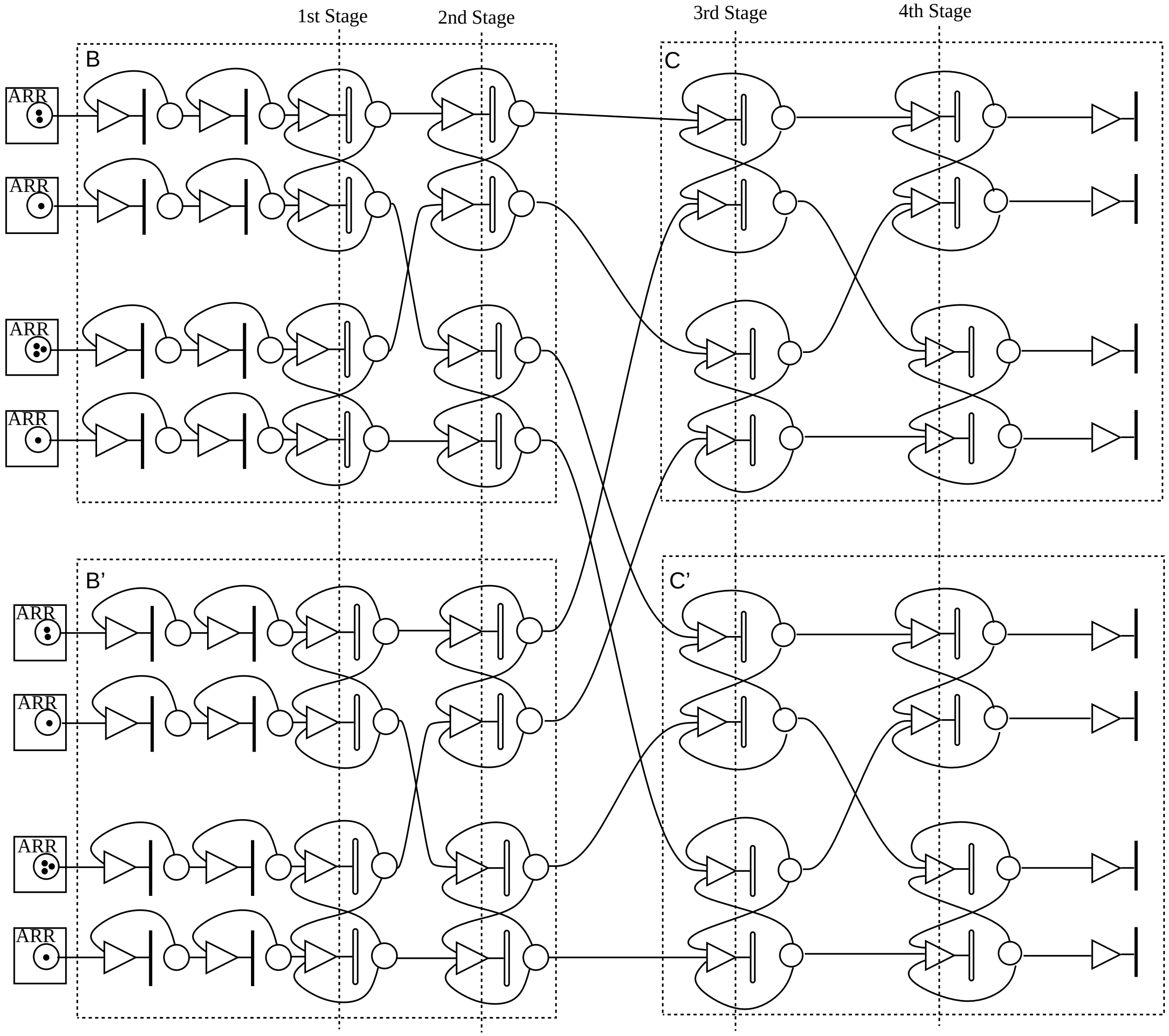


Figure 4.8: SAN representation for 8 × 8 Modified Delta Network

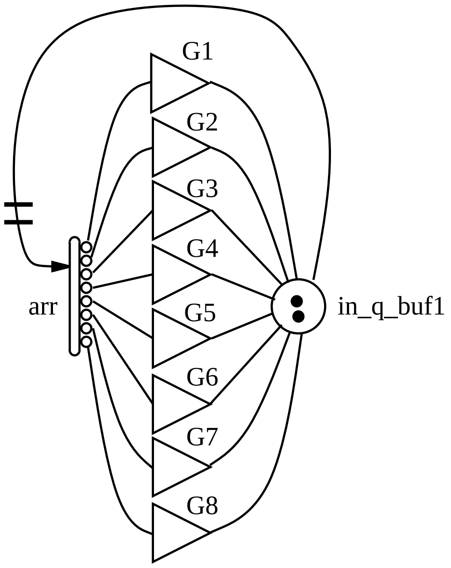


Figure 4.9: SAN for 8 × 8 Modified Delta Network ARR block

in_q_buf3_2, *in_q_buf3_3* and *in_q_buf3_4*. If the token numbers in these places are between 1 and 4, the packets are forwarded to place in_buf_1 and in_buf_3. If the token numbers in these place are between 5 and 8, the packets are forwarded to place in_buf_2 and in_buf_4. The switching elements in the second stage perform a search to find a first available route, since each packet has more than one route to reach its destination. A searching algorithm is adopted here. The input gate of the timed activity *processing_i* (*ck_1*, *ck_2*, *ck_3* or *ck_4)*, search for an available path to the next stage of switching element. Refer to the gate predicates for these gates in Tables 4.9 and 4.10 for the implementation. If an available place is found, the tokens in *in_buf_1*, *in_buf_2*, *in_buf_3* and *in_buf_4* are removed and same number of tokens are placed in the available place *in_buf_5*, *in_buf_6*, *in_buf_7* or *in_buf_8*. The activity parameters for this SAN are listed in Table 4.11. Part B' in Figure 4.8 is the same as Part B. The SAN for this submodel can be found in Figure 4.5, which is part B of the baseline banyan model. The gate functions and predicates are exactly the same as in that model. In general, the gate functions' predicates check for the number of the tokens (address) in the place of current stage (input buffer) and the availability of the designated

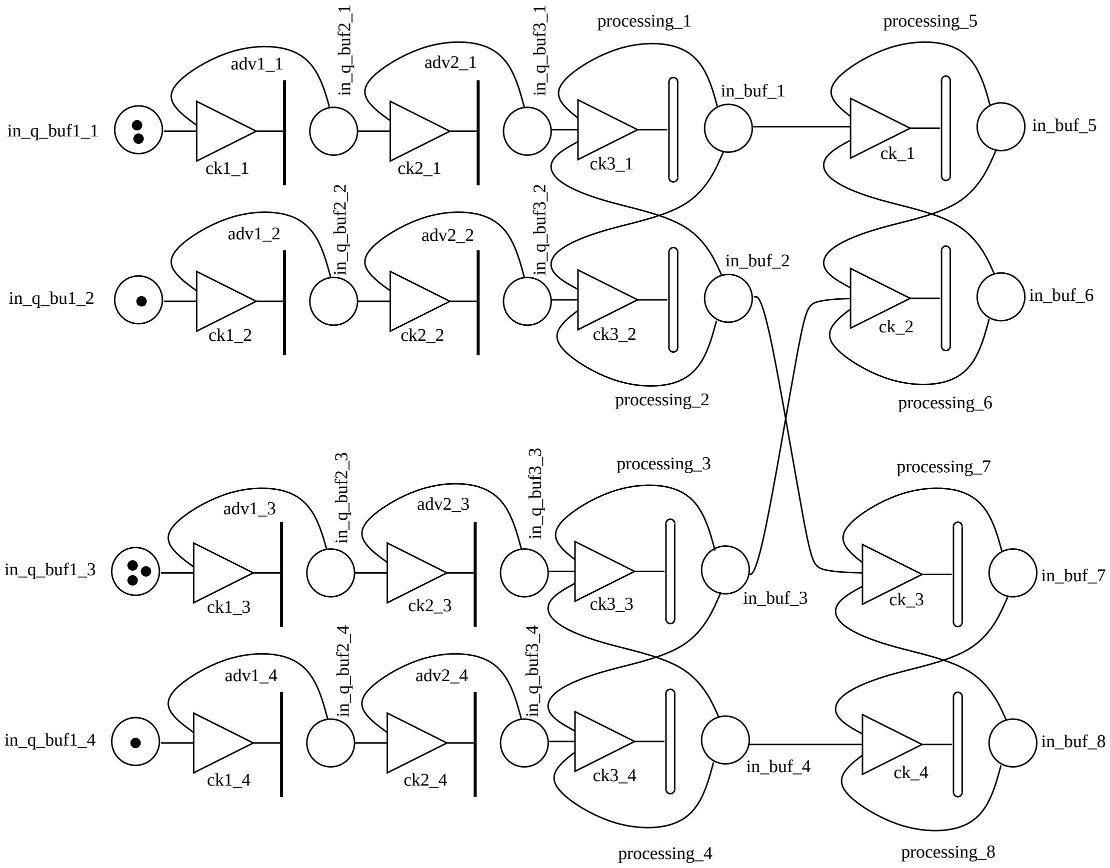


Figure 4.10: Detailed Diagram of Part B in 8 × 8 Modified Delta Network

place of the next stage (output buffer). Eventually, a packet reaches the output of last (fourth) stage switching element, represented by the place *output_i*. Functions of input gates $G_a \ldots G_d$ then subtract one token (representing one packet) each from the internal counter *packet_in_system* for every completed activity, *pkt_exit_i*. Part C' is the same as part C. The difference lies in the input gates of third stage and fourth stage switching elements. The predicates now check for packet number between 5 and 8 instead of between 1 and 4.

4.1.4 Model of 8×8 Modified Network with Multiplexer and Demultiplexers

A way of reducing the cost of the switch would be to make use of the components such as multiplexer and demultiplexer as suggested by [15]. They are usually less expensive than the switch fabrication. In addition, it is easy to increase the size of the switch in Mux-Demux design. As number of input and output ports increases, the Mux-Demux design needs only linearly increasing the number of switching elements. Whereas in other designs with interconnection networks, the number of switching elements required is increased in proportional to $(NlogN)$ as switch grows. Figure 4.11 depicts a design of 8×8 switching fabric. It consists of multiplexers, demultiplexers, and a less amount of 2×2 switching elements. As shown in the figure, the input and output ports of the switching fabric can be divided into two groups: the banyan side and the mux-demux side. This is easily observed because of the non-symmetrical configuration of the switch. We can expect a different performances for two groups of I/O ports. Due to the time delay introduced by the multiplexer, we can further expect that the throughput from banyan set of outputs will be higher than it from the mux-demux set. Thus this design is well suit to adapting the non-uniform traffic. Specifically, the switch is good in handling the non-uniform workload in which the traffic from one side (banyan side) is much heavier than the traffic from the other side (mux-demux side).

Table 4.9: Input Gate Parameters for Part B of 8 × 8 Modified Delta Switch

Gate	**Enabling Predicate**	**Function**
$ck1\_1$	MARK($in\_q\_buf1\_1$) > 0 and MARK($in\_q\_buf2\_1$) $== 0$	MARK($in\_q\_buf2\_1$)= MARK($in\_q\_buf1\_1$); MARK($in\_q\_buf1\_1$)=0;
$ck1\_2$	MARK($in\_q\_buf1\_2$) > 0 and MARK($in\_q\_buf2\_2$) $== 0$	MARK($in\_q\_buf2\_2$)= MARK($in\_q\_buf1\_2$); MARK($in\_q\_buf1\_2$)=0;
$ck2\_1$	MARK($in\_q\_buf2\_1$) > 0 and MARK($in\_q\_buf3\_1$) $== 0$	MARK($in\_q\_buf3\_1$)= MARK($in\_q\_buf2\_1$); MARK($in\_q\_buf2\_1$)=0;
$ck2\_2$	MARK($in\_q\_buf2\_2$) > 0 and MARK($in\_q\_buf3\_2$) $== 0$	MARK($in\_q\_buf3\_2$)= MARK($in\_q\_buf2\_2$); MARK($in\_q\_buf2\_2$)=0;

$ck3\_1$	($1 \leq$ MARK($in\_q\_buf3\_1$) ≤ 4 and
	MARK(in_buf_1)==0) or
	($5 \leq$ MARK($in\_q\_buf3\_1$) ≤ 8 and
	MARK(in_buf_2) $==$ 0)
		if ($1 \leq$ MARK($in\_q\_buf3\_1$) ≤ 4)
		MARK(in_buf_1)= MARK($in\_q\_buf3\_1$);
		MARK($in\_q\_buf3\_1$) = 0;
		else
		MARK(in_buf_2) = MARK($in\_q\_buf3\_1$);
		MARK($in\_q\_buf3\_1$) = 0;

$ck3\_2$	($1 \leq$ MARK($in\_q\_buf3\_2$) ≤ 4 and
	MARK(in_buf_1)==0) or
	($5 \leq$ MARK($in\_q\_buf3\_2$) ≤ 8 and
	MARK(in_buf_2) $==$ 0)
		if ($1 \leq$ MARK($in\_q\_buf3\_2$) ≤ 4)
		MARK(in_buf_1)= MARK($in\_q\_buf3\_2$);
		MARK($in\_q\_buf3\_2$) = 0;
		else
		MARK(in_buf_2) = MARK($in\_q\_buf3\_2$);
		MARK($in\_q\_buf3\_2$) = 0;

ck_1	(MARK(in_buf_1) > 0 and
	MARK(in_buf_5)==0) or
	(MARK(in_buf_1) > 0 and
	MARK(in_buf_6) $==$ 0)
		if (MARK(in_buf_5) $==$ 0)
		MARK(in_buf_5)= MARK(in_buf_1);
		MARK(in_buf_1) = 0;
		else
		MARK(in_buf_6) = MARK(in_buf_1);
		MARK(in_buf_3) = 0;

ck_2	(MARK(in_buf_3) > 0 and
	MARK(in_buf_5)==0) or
	(MARK(in_buf_3) > 0 and
	MARK(in_buf_6) $==$ 0)
		if (MARK(in_buf_5) $==$ 0)
		MARK(in_buf_5)= MARK(in_buf_3);
		MARK(in_buf_1) = 0;
		else
		MARK(in_buf_6) = MARK(in_buf_3);
		MARK(in_buf_3) = 0;

Table 4.10: Input Gate Parameters for Part B of 8 × 8 Modified Delta Switch (cont.)

Gate	**Enabling Predicate**	**Function**
$ck1\_3$	MARK($in\_q\_buf1\_3$) $>$ 0 and MARK($in\_q\_buf2\_3$) == 0	MARK($in\_q\_buf2\_3$)= MARK($in\_q\_buf1\_3$); MARK($in\_q\_buf1\_3$)=0;
$ck1\_4$	MARK($in\_q\_buf1\_4$) $>$ 0 and MARK($in\_q\_buf2\_4$) == 0	MARK($in\_q\_buf2\_4$)= MARK($in\_q\_buf1\_4$); MARK($in\_q\_buf1\_4$)=0;
$ck2\_3$	MARK($in\_q\_buf2\_3$) $>$ 0 and MARK($in\_q\_buf3\_3$) == 0	MARK($in\_q\_buf3\_3$)= MARK($in\_q\_buf2\_3$); MARK($in\_q\_buf2\_3$)=0;
$ck2\_4$	MARK($in\_q\_buf2\_4$) $>$ 0 and MARK($in\_q\_buf3\_4$) == 0	MARK($in\_q\_buf3\_4$)= MARK($in\_q\_buf2\_4$); MARK($in\_q\_buf2\_4$)=0;

$ck3\_3$	(1 $\leq$ MARK($in\_q\_buf3\_3$) $\leq$ 4 and
	MARK(in_buf_3)==0) or
	(5 $\leq$ MARK($in\_q\_buf3\_3$) $\leq$ 8 and
	MARK(in_buf_4) == 0)
		if (1 $\leq$ MARK($in\_q\_buf3\_3$) $\leq$ 4)
		MARK(in_buf_3)= MARK($in\_q\_buf3\_3$);
		MARK($in\_q\_buf3\_3$) = 0;
		else
		MARK(in_buf_4) = MARK($in\_q\_buf3\_3$);
		MARK($in\_q\_buf3\_3$) = 0;

$ck3\_4$	(1 $\leq$ MARK($in\_q\_buf3\_4$) $\leq$ 4 and
	MARK(in_buf_3)==0) or
	(5 $\leq$ MARK($in\_q\_buf3\_4$) $\leq$ 8 and
	MARK(in_buf_4) == 0)
		if (1 $\leq$ MARK($in\_q\_buf3\_4$) $\leq$ 4)
		MARK(in_buf_3)= MARK($in\_q\_buf3\_4$);
		MARK($in\_q\_buf3\_4$) = 0;
		else
		MARK(in_buf_4) = MARK($in\_q\_buf3\_4$);
		MARK($in\_q\_buf3\_4$) = 0;

ck_3	(MARK(in_buf_2) $>$ 0 and
	MARK(in_buf_7)==0) or
	(MARK(in_buf_2) $>$ 0 and
	MARK(in_buf_8) == 0)
		if (MARK(in_buf_7) == 0)
		MARK(in_buf_7)= MARK(in_buf_2);
		MARK(in_buf_2) = 0;
		else
		MARK(in_buf_8) = MARK(in_buf_2);
		MARK(in_buf_2) = 0;

ck_4	(MARK(in_buf_4) $>$ 0 and
	MARK(in_buf_7)==0) or
	(MARK(in_buf_4) $>$ 0 and
	MARK(in_buf_8) == 0)
		if (MARK(in_buf_7) == 0)
		MARK(in_buf_7)= MARK(in_buf_4);
		MARK(in_buf_4) = 0;
		else
		MARK(in_buf_8) = MARK(in_buf_4);
		MARK(in_buf_4) = 0;

Table 4.11: Activity Parameters for Part B of Modified Delta Network

Activity	**Rate**	**Probability**
*adv*1_1	inst	1
*adv*2_1	inst	1
*adv*1_2	inst	1
*adv*2_2	inst	1
*adv*1_3	inst	1
*adv*2_3	inst	1
*adv*1_4	inst	1
*adv*2_4	inst	1
*processing*_1	normal(μ)(δ)	1
*processing*_2	normal(μ)(δ)	1
*processing*_3	normal(μ)(δ)	1
*processing*_4	normal(μ)(δ)	1
*processing*_5	normal(μ)(δ)	1
*processing*_6	normal(μ)(δ)	1
*processing*_7	normal(μ)(δ)	1
*processing*_8	normal(μ)(δ)	1

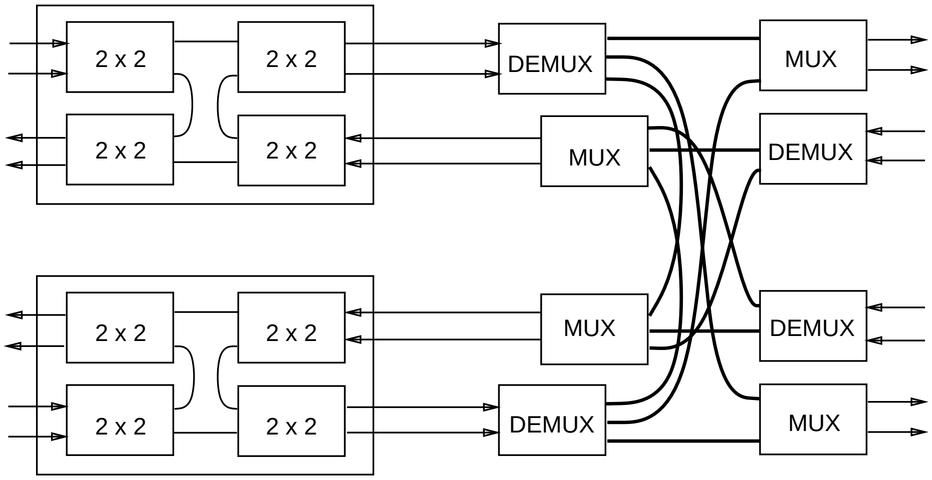


Figure 4.11: 8 × 8 Network with Mux and Demux

Packets wait their turns in the input queue in the order of their arrivals. When available, the input port controller removes the head of the queue and sends it into the switch fabric. Each switching element has a single-buffer in its input port to temperately store one packet. Each demultiplexer is connected to two or three multiplexer in order to route packets to the proper output port. At the banyan side of the network, packet is stored temporarily in the buffer of switching element. It then is sent to proper port depending on its destination. If the destination port is on the mux-demux side of the network, the packet is forwarded to one of the two demultiplexers. The function of the demultiplexer is to direct the packet to the correct path. Packets going through demultiplexer are then routed to the multiplexer. Multiplexers collect the packets to same outputs and send them to their destinations. Packets enter from mux-demux side of the switch follow similar process except they are routed by the demultiplexers and multiplexers first. Each multiplexer has a buffer which would hold 5 packets. No buffer is necessary for the demultiplexer since the speed of the lines connecting the multiplexer and demultiplexer is the same as it in the other part of the switching network. Multiple paths can be accessible to route the packet. In this case, packets are always routed to the first available route.

Figure 4.12 shows the SAN representation of this modified 8 × 8 switching network. Packet arrival is modeled by a timed activity *arr*, shown in Figure 4.13. The distribution of arrival is exponential. Three cases are associated to the arrival activity: case 1 connected to output gate *G1*, case 2 to gate *G2* and case 3 to gate *G3*. Case 1 represents that the arriving packet is designated to the output in the same 4 × 4 switch. Case 2 represents that the arriving packet is addressed to the output at the banyan side of the 8 × 8 switch but not in the same 4 × 4 switch. The arrival of packets to be routed to the mux-demux side of the switch are represented by case 3. If case 1 is chosen, gate *G1* puts 1 token in the place *in_q_buf1_i*. If case 2 is chosen, *G2* creates 2 tokens in *in_q_buf1_i*. Similarly, *G3* generates 3 tokens in *in_q_buf1_i* if case 3 is selected. Each output gate sends one token to an internal counter *packet_in_system* when executing the gate function. The

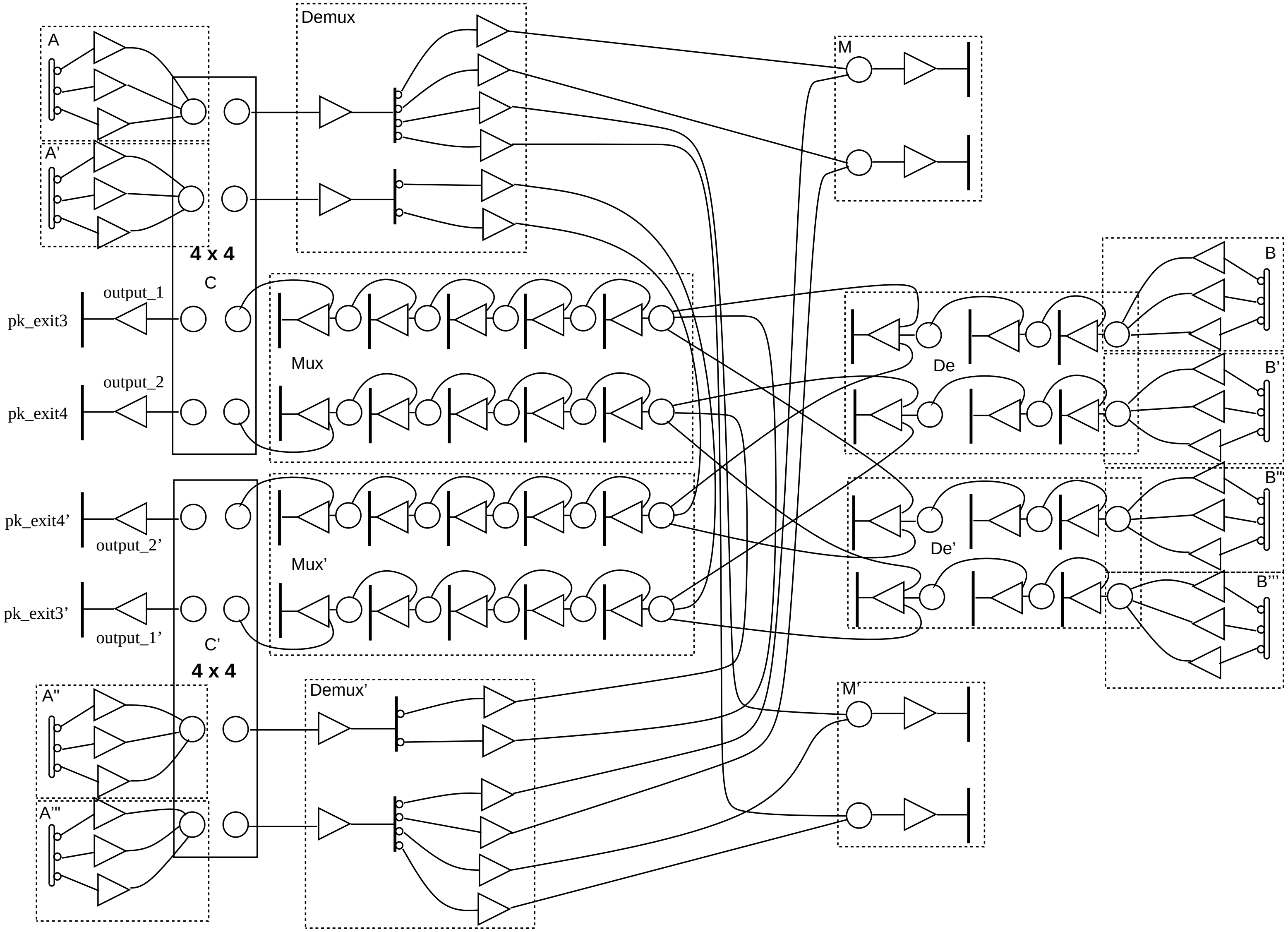


Figure 4.12: SAN representation for Modified 8 × 8 switch with Mux and Demux

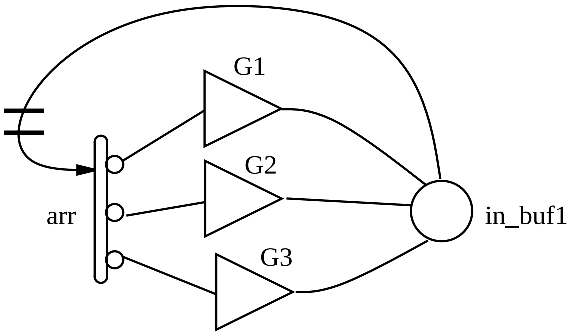


Figure 4.13: SAN Representation for Arrival in Modified 8 × 8 switch with Mux and Demux

activity parameters and output gate functions are listed in Table 4.12 and Table 4.13. Parts A, A', A" and A"' as well as parts B, B', B" and B"' in Figure 4.12 are all the same as in Figure 4.9 and represent the arrivals of packets to the switch. A SAN model for the 4 × 4 part(denoted by C in Figure 4.12) of the network is shown in Figure 4.14. Places *in_buf1_i*, *in_buf2_i* and *in_buf3_i* are the buffers in the input queue. Activities *adv1_i*, *adv2_i* and *adv3_i* are instantaneous activities which model the advance of packets in the queue. Input gates *ck1_i* and *ck2_i* examine the contents of buffers, as described in the previous models, and move the tokens from one place to the next. Blocks X, X', X" and Y are the 2 × 2 blocks shown in the block diagram (Figure 4.11). Each 2 × 2 switching element performs similar functions except the one in block Y. Of course, the gate functions in these similar 2 × 2 switch models check for different number of tokens. I will describe block X here.

This SAN model of 2 × 2 switching element includes four places, two input gates and two timed activities. Places *in_buf3_1* and *in_buf3_2* represent the input buffers of the switching element. Each holds one packet. Input gates *ck3_1* and *ck3_2* first check the type of packets in the buffers. If the packet is destined for a local output port (in this case, places *out_1* or *out_2*, and token number = 1) and the lower place *in_buf_4* is free, then the timed activity (*processing_1* or *processing_2*) is completed, the place *in_buf3_i* is emptied, and same number of tokens are added to *in_buf_4*. This signified the routing of packet. Otherwise, if the type of packets is not for the local outputs

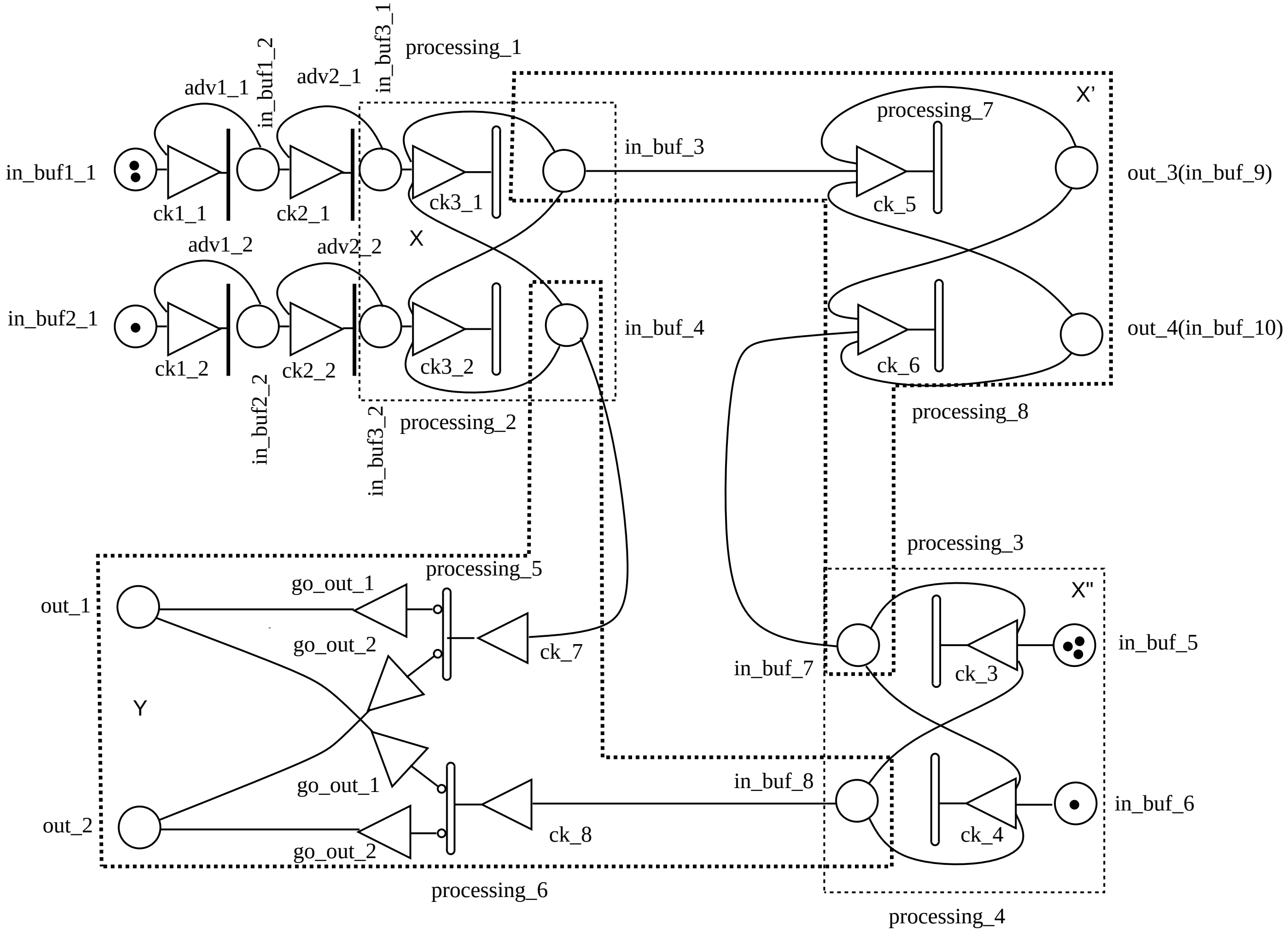


Figure 4.14: SAN representation for 4 × 4 switch in Modified 8 × 8 switch with Mux and Demux

Table 4.12: Activity Parameters for ARR block of 8 × 8 Network with Mux and Demux

Activity	**Rate**	**Probability**		
		case		
		1	2	3
arr	exp(λ)	0.3333	0.3333	0.3334

and the *in_buf_3* is free, the tokens is moved to *in_buf_3*. The gates *ck_5* and *ck_6* in X' check the packets to see which output ports they are going. If the number of tokens in the input-buffer place is 2, the packet is to be routed to the lower 4 × 4 switch. If the token number is 3, the packet is to be routed to the output ports at mux-demux side. The gates *ck_3* and *ck_4* in block X" examine the token numbers in places *in_buf_5* and *in_buf_6*. If token number is 1, the packet will be moved to place *in_buf_8*. If token number equals 2 or 3, the packet will be moved to place *in_buf_7*.

Block Y also represents a 2 × 2 switching element. Input gates *ck_7* and *ck_8* check for the existence of tokens in input-buffer places *in_buf_4* and *in_buf_8*. If true, the activity *processing_5* or *processing_6* is completed. Two cases associated with the activity. These are to signify that packets go to both outputs evenly. Therefore, one packet has 0.5 probability to choose case 1. This causes the packet to exit through place *out_1*. There is 0.5 probability that the packet will choose case 2 and exit through place *out_2*. The input gate functions and predicates for this 4 × 4 SAN model are shown in Table 4.14 and Table 4.15, respectively. The output gate functions are listed in Table 4.16 and activity attributes for the model are in Table 4.17. Part C' of Figure 4.12 is a mirror image of Part C. Everything in Part C' is the same as in Part C except that:

a) Gates *ck3_i*, *ck_3* and *ck_4* now check for token number equal 2 for the local output ports and token number 1 and 3 for the upper 4 × 4 switch and mux-demux side.

b) Gates *ck_5* and *ck_6* check for token number 1 to be routed to place *out_4*.

Table 4.13: Output Gate Parameters for ARR block of 8 × 8 with Mux and Demux

Gate	**Function**
$G1$	MARK(in_q_buf1) = 1; MARK($packet\_in\_system$) = MARK($packet\_in\_system$) + 1;
$G2$	MARK(in_q_buf1) = 2; MARK($packet\_in\_system$) = MARK($packet\_in\_system$) + 1;
$G3$	MARK(in_q_buf1) = 3; MARK($packet\_in\_system$) = MARK($packet\_in\_system$) + 1;

Table 4.14: Input Gate Parameters for Part C of 8 × 8 Switch with Mux and Demux

Gate	**Enabling Predicate**	**Function**
$ck1\_1$	MARK($in\_q\_buf1\_1$) > 0 and MARK($in\_q\_buf2\_1$) == 0	MARK($in\_q\_buf2\_1$)= MARK($in\_q\_buf1\_1$); MARK($in\_q\_buf1\_1$)=0;
$ck1\_2$	MARK($in\_q\_buf1\_2$) > 0 and MARK($in\_q\_buf2\_2$) == 0	MARK($in\_q\_buf2\_2$)= MARK($in\_q\_buf1\_2$); MARK($in\_q\_buf1\_2$)=0;
$ck2\_1$	MARK($in\_q\_buf2\_1$) > 0 and MARK($in\_q\_buf3\_1$) == 0	MARK($in\_q\_buf3\_1$)= MARK($in\_q\_buf2\_1$); MARK($in\_q\_buf2\_1$)=0;
$ck2\_2$	MARK($in\_q\_buf2\_2$) > 0 and MARK($in\_q\_buf3\_2$) == 0	MARK($in\_q\_buf3\_2$)= MARK($in\_q\_buf2\_2$); MARK($in\_q\_buf2\_2$)=0;

$ck3\_1$	(2 ≤ MARK($in\_q\_buf3\_1$) ≤ 3 and
	MARK(in_buf_3)==0) or
	(MARK($in\_q\_buf3\_1$) == 1 and
	MARK(in_buf_4) == 0)
		if (2 ≤ MARK($in\_q\_buf3\_1$) ≤ 3)
		MARK(in_buf_3)= MARK($in\_q\_buf3\_1$);
		MARK($in\_q\_buf3\_1$) = 0;
		else
		MARK(in_buf_4) = MARK($in\_q\_buf3\_1$);
		MARK($in\_q\_buf3\_1$) = 0;

$ck3\_2$	(2 ≤ MARK($in\_q\_buf3\_2$) ≤ 3 and
	MARK(in_buf_3)==0) or
	(MARK($in\_q\_buf3\_1$) == 1 and
	MARK(in_buf_4) == 0)
		if (2 ≤ MARK($in\_q\_buf3\_2$) ≤ 3)
		MARK(in_buf_3)= MARK($in\_q\_buf3\_2$);
		MARK($in\_q\_buf3\_2$) = 0;
		else
		MARK(in_buf_4) = MARK($in\_q\_buf3\_2$);
		MARK($in\_q\_buf3\_2$) = 0;

ck_3	(2 ≤ MARK(in_buf_5) ≤ 3 and
	MARK(in_buf_7)==0) or
	(MARK(in_buf_5) == 1 and
	MARK(in_buf_8) == 0)
		if (2 ≤ MARK(in_buf_5) ≤ 3)
		MARK(in_buf_7)= MARK(in_buf_5);
		MARK(in_buf_5) = 0;
		else
		MARK(in_buf_8) = MARK(in_buf_5);
		MARK(in_buf_5) = 0;

ck_4	(2 ≤ MARK(in_buf_6) ≤ 3 and
	MARK(in_buf_7)==0) or
	(MARK(in_buf_6) == 1 and
	MARK(in_buf_8) == 0)
		if (2 ≤ MARK(in_buf_6) ≤ 3)
		MARK(in_buf_7)= MARK(in_buf_6);
		MARK(in_buf_6) = 0;
		else
		MARK(in_buf_8) = MARK(in_buf_6);
		MARK(in_buf_6) = 0;

Table 4.15: Input Gate Parameters for Part C of 8 × 8 Switch with Mux and Demux (cont.)

Gate	**Enabling Predicate**	**Function**

ck_5	(MARK(in_buf_3) == 3 and
	MARK(in_buf_9)==0) or
	(MARK(in_buf_3) == 2 and
	MARK(in_buf_10) == 0)
		if (MARK(in_buf_3) == 3)
		MARK(in_buf_9)= MARK(in_buf_3);
		MARK(in_buf_3) = 0;
		else
		MARK(in_buf_10) = MARK(in_buf_3);
		MARK(in_buf_3) = 0;

ck_6	(MARK(in_buf_7) == 3 and
	MARK(in_buf_9)==0) or
	(MARK(in_buf_7) == 2 and
	MARK(in_buf_10) == 0)
		if (MARK(in_buf_7) == 3)
		MARK(in_buf_9)= MARK(in_buf_7);
		MARK(in_buf_7) = 0;
		else
		MARK(in_buf_10) = MARK(in_buf_7);
		MARK(in_buf_7) = 0;

ck_7	MARK(in_buf_4) > 0	MARK(in_buf_4) = 0;
ck_8	MARK(in_buf_8) > 0	MARK(in_buf_8) = 0;

Table 4.16: Output Gate Parameters for Part C of 8 × 8 with Mux and Demux

Gate	**Function**
go_out_1	MARK(out_1) = MARK(out_1) + 1;
go_out_2	MARK(out_2) = MARK(out_2) + 1;

Table 4.17: Activity Parameters for Part C of 8 × 8 Network with Mux and Demux

Activity	**Rate**	**Probability**	
		case 1	case 2
$adv1\_1$	inst	1	-
$adv2\_1$	inst	1	-
$adv1\_2$	inst	1	-
$adv2\_2$	inst	1	-
$processing\_1$	normal(μ)(δ)	1	-
$processing\_2$	normal(μ)(δ)	1	-
$processing\_3$	normal(μ)(δ)	1	-
$processing\_4$	normal(μ)(δ)	1	-
$processing\_5$	normal(μ)(δ)	1	-
$processing\_6$	normal(μ)(δ)	1	-
$processing\_7$	normal(μ)(δ)	0.5	0.5
$processing\_8$	normal(μ)(δ)	0.5	0.5

Two SAN models of demultiplexers are shown in Figure 4.15 and Figure 4.16. Figure 4.15, a detailed depiction of part Demux in the 8 × 8 switch, has two places, two activities, two input gates and six output gates. Places *in_buf_9* and *in_buf_10* are input buffers of the demultiplexers. These input buffers are also the output buffers *out_3* and *out_4* in part C of the switch. Input gates *Gdemux_1* and *Gdemux_2* check for the presence of tokens in input-buffer places. If they hold, the instantaneous activity *Ademux_1* or *Ademux_2* is enabled and completed immediately. Upon completion, one case of the activity will be chosen. There are four cases, *Ga, Gb, Gc* and *Gd*, associated with *Ademux_1* with equal probability (0.25) and two cases, *Ge* and *Gf*, associated with *Ademux_2* also with equal probability (0.5). This is to make sure that one packet bears the equivalent probability to each output. If the packets is designated to the output ports of the lower 4×4 switch (token number = 2), the packet is routed to the multiplexer in part Mux'. Otherwise, the packets, after completing the activity *Ademux_1* and choosing one of the four cases, go to multiplexer in part M or M'. Part Demux' in Figure 4.12 is a mirror image of SAN model in Demux. Every SAN entity in Demux' is the same as in Demux.

Another demultiplexer model is shown in Figure 4.16. This is the SAN for part De and De' in the 8×8 model. It consists an input queue of 3 buffers, with activities *adv*'s and associated input gates *ck1*'s and *ck2*'s. This is exactly the same as the input queue mentioned above. Input gates *Gdemux_3* and *Gdemux_4* check the address of the packet (number of tokens) in place *in_buf3_3* and *in_buf3_4*. If the token number equals to 1 or 3, the packet will be routed to place *mux_buf1_1* in part Mux. If the token number is 2 or 3, the packet will be sent to place *mux_buf1_2'* in part Mux'. The input gate, output gate and activity parameters of both demultiplexer are listed in Tables 4.18, 4.19 and 4.20.

Figure 4.17 and Figure 4.18 show two SAN models for multiplexers. One of them, in Figure 4.17, consisting of two queues of five buffers each is the detailed drawing for part Mux in Figure 4.12. The SAN of queue is the same as input queue at each input port. It has five places representing five

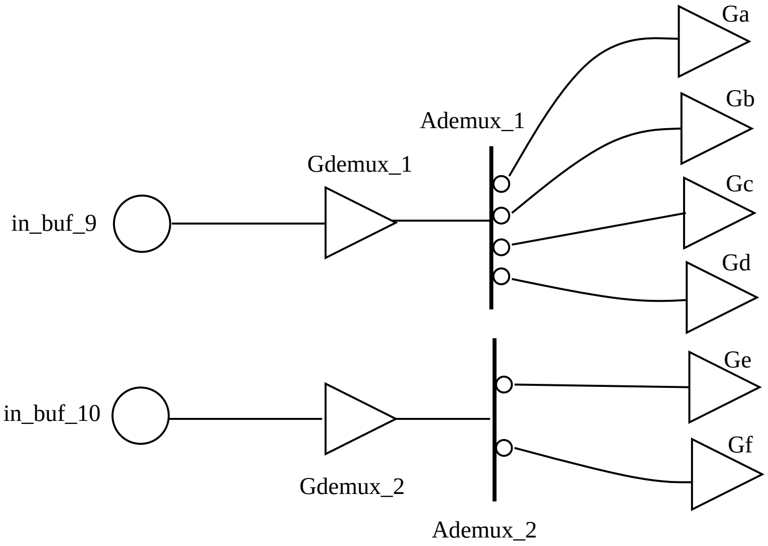


Figure 4.15: SAN representation for 1st Demux in Modified 8 × 8 switch with Mux and Demux

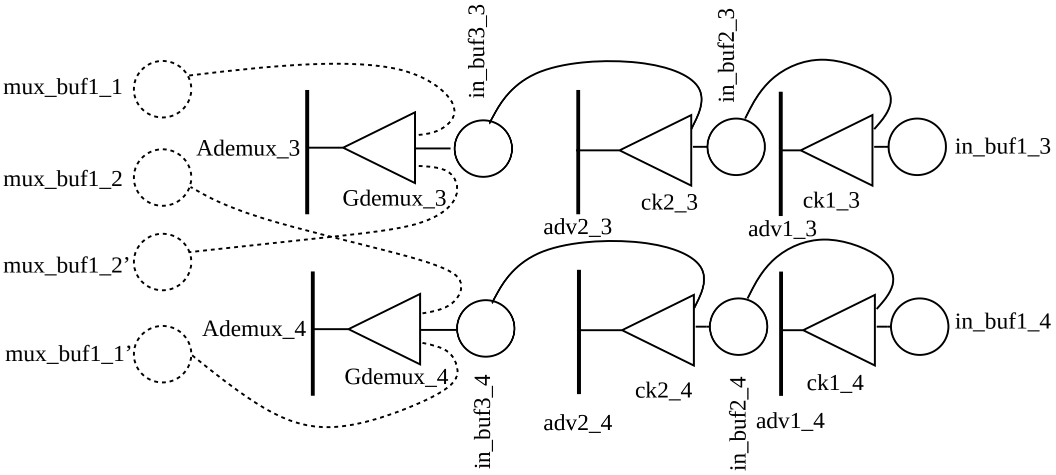


Figure 4.16: SAN representation for 2nd Demux in Modified 8 × 8 switch with Mux and Demux

Table 4.18: Input Gate Parameters for Demux of 8 × 8 Switch with Mux and Demux

Gate	**Enabling Predicate**	**Function**
$Gdemux\_1$	MARK(in_buf_9) $>$ 0	MARK(in_buf_9) = 0;
$Gdemux\_2$	MARK(in_buf_10) $>$ 0	MARK(in_buf_10) = 0;
$ck1\_3$	MARK($in\_q\_buf1\_3$) $>$ 0 and MARK($in\_q\_buf2\_3$) == 0	MARK($in\_q\_buf2\_3$)= MARK($in\_q\_buf1\_3$); MARK($in\_q\_buf1\_3$)=0;
$ck1\_4$	MARK($in\_q\_buf1\_4$) $>$ 0 and MARK($in\_q\_buf2\_4$) == 0	MARK($in\_q\_buf2\_4$)= MARK($in\_q\_buf1\_4$); MARK($in\_q\_buf1\_4$)=0;
$ck2\_3$	MARK($in\_q\_buf2\_3$) $>$ 0 and MARK($in\_q\_buf3\_3$) == 0	MARK($in\_q\_buf3\_3$)= MARK($in\_q\_buf2\_3$); MARK($in\_q\_buf2\_3$)=0;
$ck2\_4$	MARK($in\_q\_buf2\_4$) $>$ 0 and MARK($in\_q\_buf3\_4$) == 0	MARK($in\_q\_buf3\_4$)= MARK($in\_q\_buf2\_4$); MARK($in\_q\_buf2\_4$)=0;

$Gdemux\_3$	(MARK($in\_q\_buf3\_3$) == (1 or 3) and
	MARK(mux_buf1_1)==0) or
	(MARK($in\_q\_buf3\_3$) == (2 or 3) and
	MARK(mux_buf1_2') == 0)
		if (MARK($in\_q\_buf3\_3$) == (1 or 3)
		MARK(mux_buf1_1)= MARK($in\_q\_buf3\_3$);
		MARK($in\_q\_buf3\_3$) = 0;
		else
		MARK(mux_buf1_2') = MARK($in\_q\_buf3\_3$);
		MARK($in\_q\_buf3\_3$) = 0;

$Gdemux\_4$	(MARK($in\_q\_buf3\_4$) == (1 or 3) and
	MARK(mux_buf1_2)==0) or
	(MARK($in\_q\_buf3\_4$) == (2 or 3) and
	MARK(mux_buf1_1') == 0)
		if (MARK($in\_q\_buf3\_4$) == (1 or 3)
		MARK(mux_buf1_2)= MARK($in\_q\_buf3\_4$);
		MARK($in\_q\_buf3\_4$) = 0;
		else
		MARK(mux_buf1_1') = MARK($in\_q\_buf3\_4$);
		MARK($in\_q\_buf3\_4$) = 0;

Table 4.19: Output Gate Parameters for Demux of 8 × 8 with Mux and Demux

Gate	**Function**
Ga	MARK(mux_buf_1) = MARK(mux_buf_1) + 1;
Gb	MARK(mux_buf_2) = MARK(mux_buf_2) + 1;
Gc	MARK(mux_buf_1') = MARK(mux_buf_1') + 1;
Gd	MARK(mux_buf_2') = MARK(mux_buf_2') + 1;
Ge	MARK(mux_buf1_1') = 2;
Gf	MARK(mux_buf1_2') = 2;

Table 4.20: Activity Parameters for Demux of 8 × 8 Network with Mux and Demux

Activity	**Rate**	**Probability**			
		case 1	case 2	case 3	case 4
$adv1\_3$	inst	1	-	-	-
$adv2\_3$	inst	1	-	-	-
$adv1\_4$	inst	1	-	-	-
$adv2\_4$	inst	1	-	-	-
$Ademux\_1$	inst	0.25	0.25	0.25	0.25
$Ademux\_2$	inst	0.5	0.5	-	-
$Ademux\_3$	inst	1	-	-	-
$Ademux\_4$	inst	1	-	-	-

buffers. These are *mux_buf_1* … *mux_buf_10*. Activities *Amux_1* … *Amux_10* are instantaneous activities signify the advance of the packets in queue. The input gates *Gmux_1* … *Gmux_10* are the gates for checking whether the next buffer in queue is free and, if free, to enable the activities and move the tokens forward. *in_buf_5* and *in_buf_6* are the output buffers of the multiplexers also serve as input buffers for the 4 × 4 switch (part C in Figure 4.12). Mux' in the 8 × 8 SAN is the same as Mux.

The second multiplexer model is shown in Figure 4.18. It is the detailed drawing for part M and M' in Figure 4.12. The SAN model has only two places, representing two single-buffers in multiplexer. *Gout1* and *Gout2* are the input gates associated with instantaneous activities *pk_exit1* and *pk_exit2*. *pk_exit1* and *pk_exit2* represent the events packets exiting the system. When completed, gate functions in *Gout1* and *Gout2* subtract one token each from the internal counter *packet_in_system*. The input gates *output_1, output_2, output_1'* and *output_2'* along with their associated activities *pk_exit3, pk_exit4, pk_exit5* and *pk_exit6* in Figure 4.12 also signify the exit of packets. Tables 4.21 and 4.22 show the input gate and activity parameters for the muxltiplexers.

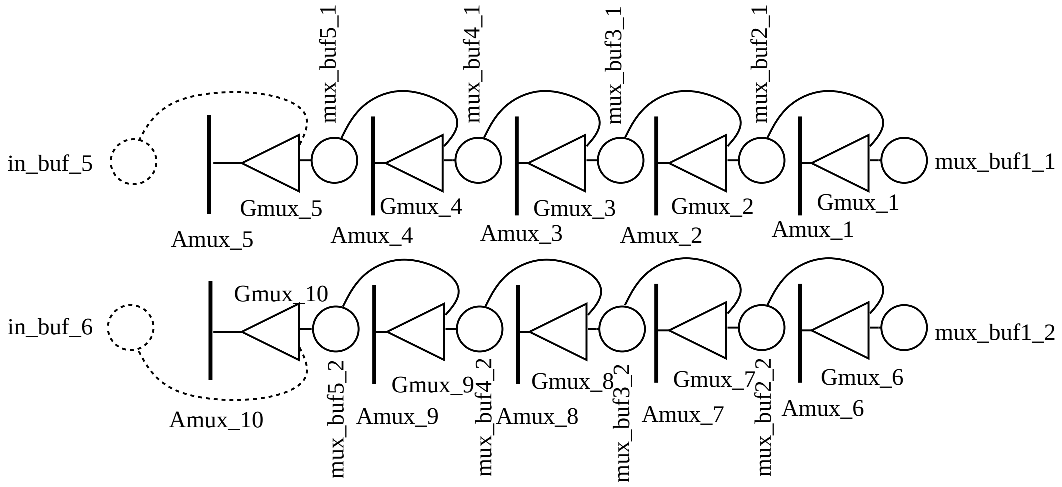


Figure 4.17: SAN representation for 1st Mux in Modified 8 × 8 switch with Mux and Demux

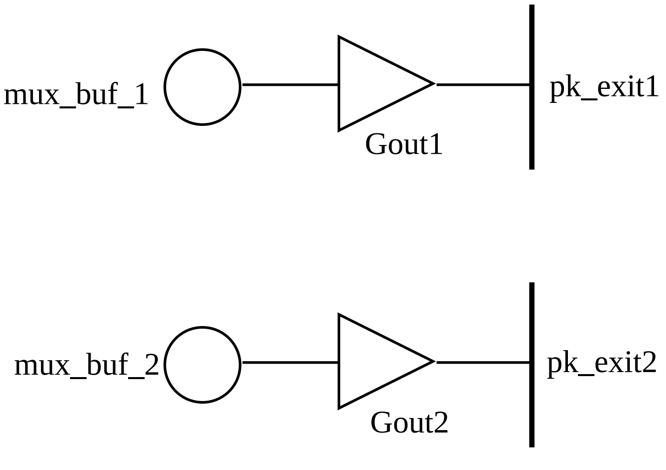


Figure 4.18: SAN representation for 2nd Mux in Modified 8 × 8 switch with Mux and Demux

Table 4.21: Input Gate Parameters for Mux's of 8 × 8 Switch with Mux and Demux

Gate	**Enabling Predicate**	**Function**
$Gmux\_1$	MARK(mux_buf1_1) > 0 and MARK(mux_buf2_1) $== 0$	MARK(mux_buf2_1)= MARK(mux_buf1_1); MARK(mux_buf1_1)=0;
$Gmux\_2$	MARK(mux_buf2_1) > 0 and MARK(mux_buf3_1) $== 0$	MARK(mux_buf3_1)= MARK(mux_buf2_1); MARK(mux_buf2_1)=0;
$Gmux\_3$	MARK(mux_buf3_1) > 0 and MARK(mux_buf4_1) $== 0$	MARK(mux_buf4_1)= MARK(mux_buf3_1); MARK(mux_buf3_1)=0;
$Gmux\_4$	MARK(mux_buf4_1) > 0 and MARK(mux_buf5_1) $== 0$	MARK(mux_buf5_1)= MARK(mux_buf4_1); MARK(mux_buf4_1)=0;
$Gmux\_5$	MARK(mux_buf5_1) > 0 and MARK(in_buf_5) $== 0$	MARK(in_buf_5)= MARK(mux_buf5_1); MARK(mux_buf5_1)=0;
$Gmux\_6$	MARK(mux_buf1_2) > 0 and MARK(mux_buf2_2) $== 0$	MARK(mux_buf2_2)= MARK(mux_buf1_2); MARK(mux_buf1_2)=0;
$Gmux\_7$	MARK(mux_buf2_2) > 0 and MARK(mux_buf3_2) $== 0$	MARK(mux_buf3_2)= MARK(mux_buf2_2); MARK(mux_buf2_2)=0;
$Gmux\_8$	MARK(mux_buf3_2) > 0 and MARK(mux_buf4_2) $== 0$	MARK(mux_buf4_2)= MARK(mux_buf3_2); MARK(mux_buf3_2)=0;
$Gmux\_9$	MARK(mux_buf4_2) > 0 and MARK(mux_buf5_2) $== 0$	MARK(mux_buf5_2)= MARK(mux_buf4_2); MARK(mux_buf4_2)=0;
$Gmux\_10$	MARK(mux_buf5_2) > 0 and MARK(in_buf_6) $== 0$	MARK(in_buf_6)= MARK(mux_buf5_2); MARK(mux_buf5_2)=0;
$Gout1$	MARK(mux_buf_1) > 0	MARK(mux_buf_1) $= 0$
$Gout2$	MARK(mux_buf_2) > 0	MARK(mux_buf_2) $= 0$

Table 4.22: Activity Parameters for Mux's of 8 × 8 network with Mux and Demux

Activity	**Rate**	**Probability**
$Amux\_1$	inst	1
$Amux\_2$	inst	1
$Amux\_3$	inst	1
$Amux\_4$	inst	1
$Amux\_5$	inst	1
$Amux\_6$	inst	1
$Amux\_7$	inst	1
$Amux\_8$	inst	1
$Amux\_9$	inst	1
$Amux\_10$	inst	1
pk_exit1	inst	1
pk_exit2	inst	1

4.2 Performance Variables

We are interested in three performance variables. They are *"expected throughput of the system"*, *"rejection probability"* and the *"expected system time delay"* for a successfully delivered packet. For those models with back–pressure algorithm, the expected throughput of the system is the *arrival rate* minus the *expected total rate of rejection*. The expected total rate of rejection can be obtained by the performance parameter *packet_loss_prob* in METASAN experiment file. Using Little's result, the total time delay can be calculated from the parameter *number_packet_in_system* following the formula:

$$E[\#\ in\ system] = \lambda \cdot TT$$

$$TT = \frac{1}{\lambda} E[\#\ in\ system]$$

where:

$E[\#\ in\ system]$= *expected number of packets in system.*

λ= *average arrival rate.*

TT= *expected time delay.*

For those models with the reject–retransmission algorithm, the performance variables can be derived from the *rate of departure* from the switch and the *expected number of packets in system.* Figure 4.19 shows a sketch of the system. Assume that a system has a rate of arrival λ. The system consists of a finite queue and a service unit. When the queue is full, the incoming customers are rejected by the system. The rate of rejection is λ_d. The service unit provides service to the customers in the queue. A customer can leave the system with completion of service or without. The rate of departure with completion of service is λ", while the rate of departure without completion of service is λ'. Customers failed to complete the service are immediately fed back to the system and start over. The rate of feedback is equal to λ'. However, a fed-back customer can be rejected due to queue full. We denote the rate of feedback rejection λ'_d. The variables we

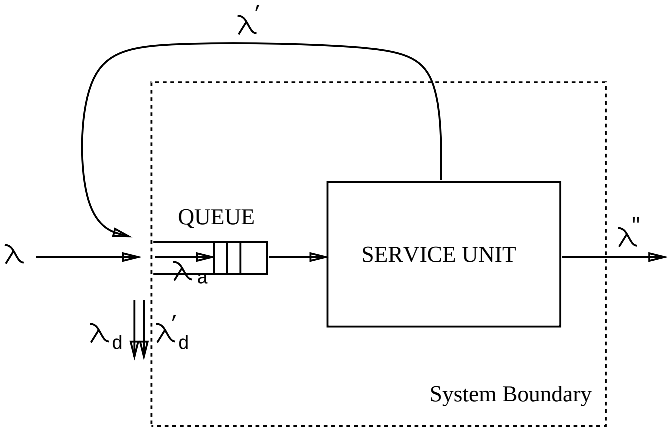


Figure 4.19: Reject-Retransmission System Diagram

need are: the *expected throughput of the system* (λ"), *expected rate of rejection* ($\lambda_d + \lambda'_d$) and the *expected time delay for a serviced customer* (TT). We denote λ_a to be the actual arrival rate seen by the system. Since the number in the system is at all times finite, the *total rate of arrival* equal to the *total rate of departure*, i.e.,

$$\lambda_a = \lambda' + \lambda"$$

Given a customer in the system, the probability for it to successfully be routed to its destination is:

$$p_s = \frac{\lambda"}{\lambda_a} = \frac{\lambda"}{\lambda' + \lambda"}$$

Thus the probability a customer in the system does not successfully reach his destination on a single try is $(1 - p_s)$. The expected number of attempts that a customer which eventually receives service makes before receiving successful service is:

$$E[N] = \sum_{i=1}^{\infty} i\ (1 - p_s)^{i-1} p_s$$

Then, by Little's result, the expected time in the switch on a single attempt is

$$E[T] = \frac{E[\#\ in\ system]}{\lambda_a}$$

and the expected system time delay for a serviced customer is:

$$TT = E[N]\,E[T] = \frac{E[\#\ in\ system]}{\lambda_a} \sum_{i=1}^{\infty} i\ (1 - p_s)^{i-1} p_s$$

The term $\sum_{i=1}^{\infty} i\ (1 - p_s)^{i-1} p_s$ has an analytical solution of $\frac{1}{p_s}$. Thus the expected system time delay for a serviced customer becomes:

$$TT = \frac{E[\#\ in\ system]}{\lambda_a\ p_s}$$

In specific, we will use the variables packet loss probability, number packet in system, and actual arrival rate to calculate the expected time delay.

4.3 Assumed Workload

Most of the studies on banyan networks assumed uniform workload to the system. In real application, this is rarely the case. Thus a switch must be able to handle non-uniform traffic at all time. Not only the arrival rate to different inputs can be different, but also the routing probably to each output may not be the same. Therefore, three types of workload are studied in our models. The first type is the uniform workload. Assuming same arrival rate for all incoming packets at different input ports. This uniform traffic load has been the assumption of several studies on banyan network [2, 10, 24, 25]. However, the assumption of uniform load does not usually reflect the realistic traffic situation in the real system. In the real system, a switch has to accommodate a wide range of bandwidths. In specific, the traffic flow from the network side is not the same as it from the local side. A switch should be capable of handling these different workloads. Hence the traffic pattern of non-uniform distribution need to be addressed. Two modes of non-uniform traffic are considered:

a) Type A: uniform total traffic to each input port but there are different distribution for different destinations.

b) Type B: non-uniform traffic to each input port as well as different distribution for different destinations.

We will do this in the next chapter, where we present the results of evaluation studies based on thesis models.

CHAPTER 5

RESULTS AND DISCUSSION

Three SAN models based on the three designs of the Multistage Interconnection Network were constructed in the previous chapter. Each SAN model was augmented to represent two different algorithms for handling the blocked packets. As mentioned in previous chapter, they are back-pressure (B-P) and reject algorithm. In the second algorithm, the blocked packet is dropped. For a fair comparison, we will consider that each packet is retransmitted by upper layer protocol, as would be the case of the reliable data transfer. We call this algorithm reject-retransmission (R-R). Each SAN model assumed in this way was evaluated under three different types of workload, as described below. As the result, a total of eighteen sets of simulation experiments are conducted. Table 5.1 lists all the combinations of simulation runs. We will discuss the simulation results by comparing them from different designs for the same type of workload.

Table 5.1: Simulation Experiment Sets

		Uniform Workload, Uniform Routing Probability	Uniform Workload, Non-uniform Routing Probability	Non-uniform Workload, Non-uniform Routing Probability
Baseline	Back-Pressure	X	X	X
	Reject-Retransmission	X	X	X
Mux–Demux	Back-Pressure	X	X	X
	Reject-Retransmission	X	X	X
Modified Delta	Back-Pressure	X	X	X
	Reject-Retransmission	X	X	X

Table 5.2: Packet Loss Probability

		Uniform Workload, Uniform Routing Prob.				
arr/proc		0.1	0.2	0.3	0.4	0.5
Baseline	B-P	.00087 ± .00015	.019 ± .0017	.114 ± .010	.256 ± .023	.386 ± .030
	R-R	.0007 ± .007	.003 ± .008	.025 ± .009	.090 ± .008	.203 ± .007
Network with Mux&Demux	B-P	.00034 ± .00006	.0057 ± .00034	.068 ± .005	.216 ± .017	.353 ± .024
	R-R	.0008 ± .007	.002 ± .007	.012 ± .007	.038 ± .007	.140 ± .005
Modified Delta	B-P	.00067 ± .00006	.0100 ± .0009	.051 ± .004	.136 ± .011	.239 ± .019
	R-R	.007 ± .008	.002 ± .009	.026 ± .009	.103 ± .008	.227 ± .007

Table 5.3: Expected Packet Time Delay

		Uniform Workload, Uniform Routing Prob.				
arr/proc		0.1	0.2	0.3	0.4	0.5
Baseline	B-P	.350 ± .001	.904 ± .007	1.890 ± .023	3.157 ± .039	4.487 ± .054
	R-R	.331 ± .027	.772 ± .060	1.412 ± .134	2.242 ± .152	3.266 ± .151
Network with Mux&Demux	B-P	.319 ± .001	.938 ± .006	3.192 ± .041	5.510 ± .059	7.579 ± .114
	R-R	.299 ± .144	.695 ± .038	1.294 ± .085	2.416 ± .162	4.126 ± .221
Modified Delta	B-P	.23104 ± .00005	.5627 ± .0039	1.074 ± .011	1.775 ± .019	2.616 ± .042
	R-R	.454 ± .042	1.032 ± .077	1.815 ± .133	2.854 ± .176	4.179 ± .018

5.1 Uniform Workload, Uniform Routing Probability

In the first group of experiments, the performance of each switch design was studied under uniform workload and uniform routing probability. With the arrival rate set to one packet per unit time at each input port and equal probability of routing each packet to a given output port, we observed the behavior of three designs of switch using the two algorithms handling the blocked packets. The packet loss probability and expected delays for a serviced packet obtained for various scenarios are given in Table 5.2 and Table 5.3, respectively. All values reported in this section are at 95% level of confidence. Figure 5.1 gives the *packet loss probability* of switch models under the back-pressure algorithm. Figure 5.2 shows the *expected time delay* of these switch models. Both figures show the simulation results as the processing time of switch element is varied from 0.1 to 0.5 time unit, while the arrival rate is held constant at 1.

Figure 5.1: Packet Loss Probability of Switch Models at Uniform Workload, Uniform Routing Probability (Back-Pressure)

Figure 5.2: Expected Time Delay of Switch Models at Uniform Workload, Uniform Routing Probability (Back-Pressure)

It can be seen that the packet loss probability for all three models is relatively small (< 0.02) when the processing rate is 5–10 times faster than the arrival rate. The packet loss probability starts to increase when the ratio of arrival rate and processing rate passes 0.20. The rates of increase (slopes of the curves) are similar for modles of basic banyan and Mux–Demux. However, the model for Modified 8×8 delta network has the smallest slope for its curve and the lowest packet loss probability of all. The basic banyan 8×8 network, with three-stage switching elements and single path, has the highest packet loss probability. As processing rate slows down to 0.7 of the arrival rate, the simulation of model shows that basic banyan 8×8 network reaches a packet loss probability of 0.55.

The expected packet time delay of each switch design is plotted in Figure 5.2. We note that the model for modified 8×8 delta network has the shortest expected packet time delay and the model for basic banyan 8×8 network has the longest delayat all utilizations. This is probably due to the multiple path characteristic of modified 8×8 delta network, by which more packets can be routed and thus reduce the waiting time in queue. The blocked packet handling algorithm also affect the results. In the simulation runs with back-pressure algorithm, the packets are held in the previous switching element until current switching element is free. This would cause time delay of a packet to occur within the switch (in the buffers of the switching element) rather than in the input queues.

Figure 5.3 and Figure 5.4 are the packet loss probability and expected time delay of three switch models with reject-retransmission algorithm. In Figure 5.3, 8×8 network with multiplexer and demultiplexer performs the best of the three at the high $arrival/processing$ ratio (> 0.30). When the ratio is lower than 0.30, the model of the design with modified 8×8 delta network seems to perform the best. It is interesting to note, however, that packet loss probability of the model for modified 8×8 delta network becomes the highest of the three models once the $arrival/processing$

ratio reaches about 0.35 and stays as the highest afterward. It will reject 22.6% of the incoming packets at the input queue as *arrival/processing* ratio equal to 0.5.

As shown in Figure 5.4, model for modified 8×8 delta network maintains the highest expected packet time delay at all simulation experiments with reject-retransmission algorithm as well as uniform workload and routing probability. While expected packet time delay of the model for 8×8 network with multiplexer starts with the lowest of all at the low *arrival/processing* ratio, but increases to more than it is in the basic banyan 8×8 model after the ratio raises beyond 0.35. It is noted that both the packet loss probability and expected packet time delay of the model for modified 8×8 delta network are the highest of the three models at most of the time regardless of its multiple path in the design.

5.2 Uniform Workload, Non-uniform Routing Probability

The second group of simulation experiments are shown in Tables 5.4 and 5.5. Simulations were run by varying the *arrival rate/processing rate* ratio from 0.1 to 0.5 under uniform workload, non-uniform routing probability conditions. The arrival rate is kept at one packet per time unit; while the processing rate of the switching elements are changed from 10 to 2 packets per time unit. The routing probability to each of the local output ports is 0.2 and the routing probability to each of the network port is 0.05 for each incoming packet. All simulation results listed in this section are within 95% confidence interval. Figure 5.5 and Figure 5.6 depict the packet loss probability and expected packet time delay of three models with back-pressure algorithm. In Figure 5.5, we see that the packet loss probability of the model for network with multiplexer and demultiplexer smallest of the three at the low *arrival/processing* ratios (< 0.20). The packet loss probability of the modified 8×8 delta network shows medium loss of three at the small ratio and increases more slowly than that of the 8×8 network with mutiplexer and demultiplexer beyond 0.25 utilization and becomes the lowest of the three. The basic banyan network has the highest loss probability

Figure 5.3: Packet Loss Probability of Switch Models at Uniform Workload, Uniform Routing Probability (Reject-Retransmission)

Figure 5.4: Expected Time Delay of Switch Models at Uniform Workload, Uniform Routing Probability (Reject-Retransmission)

Table 5.4: Packet Loss Probability

		Uniform Workload, Non-uniform Routing Prob.				
$arr/proc$		0.1	0.2	0.3	0.4	0.5
Baseline	B-P	.0016 ± .00008	.055 ± .0026	.246 ± .011	.426 ± .018	.531 ± .023
	R-R	.001 ± .002	.003 ± .002	.663 ± .029	.224 ± .020	.366 ± .020
Network with Mux&Demux	B-P	.00345 ± .00001	.006 ± .000648	.1278 ± .0037	.2838 ± .0081	.406 ± .012
	R-R	.006 ± .003	.006 ± .003	.0199 ± .003	.063 ± .028	.188 ± .012
Modified Delta	B-P	.00094 ± .00004	.0183 ± .0008	.098 ± .004	.236 ± .011	.369 ± .017
	R-R	.006 ± .003	.002 ± .003	.088 ± .021	.272 ± .021	.406 ± .016

Table 5.5: Expected Packet Time Delay

		Uniform Workload, Non-uniform Routing Prob.				
$arr/proc$		0.1	0.2	0.3	0.4	0.5
Baseline	B-P	.378 ± .0006	1.204 ± .008	2.808 ± .020	4.651 ± .033	6.237 ± .030
	R-R	.349 ± .011	.889 ± .045	1.882 ± .092	2.278 ± .121	4.653 ± .161
Network with Mux&Demux	B-P	.3308 ± .0004	1.274 ± .016	4.040 ± .018	6.362 ± .030	8.613 ± .045
	R-R	.312 ± .013	.745 ± .039	1.474 ± .075	3.054 ± .174	5.092 ± .197
Modified Delta	B-P	.2416 ± .0003	.648 ± .002	1.392 ± .0088	2.436 ± .021	3.620 ± .029
	R-R	.476 ± .021	1.184 ± .057	2.468 ± .105	4.220 ± .095	5.739 ± .089

in all the simulation runs under this back-pressure algorithm and workload. 53% of the packets are rejected at $arrival/processing$ ratio of 0.5 for the basic banyan network.

The expected time delay of packets in model for modified 8×8 delta network is the lowest among three models, as shown in Figure 5.6. Both the 8×8 Mux–Demux and basic banyan network have expected time delays very close when arrival rate/processing rate < 0.2. The expected time delay of the basic banyan network then increases rapidly and becomes the largest of the three models. Overall, the modified 8×8 delta network performs the best under this workload scenario and contention resolution mechanism.

The results of simulation runs of three models with reject-retransmission algorithm at uniform workload and non-uniform routing probability are plotted in Figure 5.7 and Figure 5.8. In this case, the 8×8 network with multiplexer and demultiplexer has the lowest packet loss probability

Figure 5.5: Packet Loss Probability of Switch Models at Uniform Workload, Non-uniform Routing Probability (Back-Pressure)

Figure 5.6: Expected Time Delay of Switch Models at Uniform Workload, Non-uniform Routing Probability (Back-Pressure)

Table 5.6: Packet Loss Probability

		Non-uniform Workload, Non-uniform Routing Prob.				
arr/proc		0.1	0.2	0.3	0.4	0.5
Baseline	B-P	.01487 ± .00024	.2263 ± .0014	.4085 ± .0027	.5094 ± .0042	.5768 ± .0061
	R-R	.007 ± .002	.018 ± .008	.109 ± .024	.256 ± .021	.380 ± .017
8×8 w/ Mux	B-P	.00357 ± .00033	.0562 ± .0049	.1957 ± .0013	.3796 ± .0079	.503 ± .014
	R-R	.004 ± .002	.017 ± .007	.052 ± .023	.140 ± .020	.254 ± .021
Modified Delta	B-P	.00556 ± .00021	.0971 ± .0015	.2720 ± .0023	.3990 ± .0034	.4803 ± .0046
	R-R	.003 ± .003	.017 ± .003	.125 ± .022	.287 ± .020	.413 ± .017

under most workload. The model for modified 8×8 delta network has the highest loss probability. The packet loss probability is 0.40 in the modified 8×8 delta network at *arrival/processing* ratio of 0.5. The curve of model for the 8×8 network with multiplexer and demultiplexer in Figure 5.8 shows the lowest expected time delay until the utilization reaches about 0.45. After this point, the basic banyan network shows a lesser delay. Under all utilizations, the modified 8×8 delta network has the highest time delay.

5.3 Non-uniform Workload, Non-uniform Routing Probability

Table 5.6 and Table 5.7 are the packet loss probability and expected time delay of models under non-uniform workload and non-uniform routing probability. All simulation results reported in this section are at 90% level of confidence. Arrival is modeled as a Poisson process, where the arrival rate at each input port on the local side (a total of 4 ports) is 1.6 packets per unit time. The rate is 0.4 packet per unit time at input port on the network side. A total rate of 8 packets per unit time to the system. This is the same as the previous two groups. The processing rate for the switch varies from 10 to 2 packets per unit time.

A comparison of three models with back-pressure algorithm is shown in Figure 5.9 and Figure 5.10. As can be seen in Figure 5.9, the packet loss probability of model for the basic banyan network is the highest of the three. It rejects 57% of the incoming packets when *arrival rate/processing rate*

Figure 5.7: Packet Loss Probability of Switch Models at Uniform Workload, Non-uniform Routing Probability (Reject-Retransmission)

Figure 5.8: Expected Time Delay of Switch Models at Uniform Workload, Non-uniform Routing Probability (Reject-Retransmission)

Table 5.7: Expected Packet Time Delay

		Non-uniform Workload, Non-uniform Routing Prob.				
arr/proc		0.1	0.2	0.3	0.4	0.5
Baseline	B-P	.4562 ± .0009	1.633 ± .005	2.837 ± .017	3.970 ± .045	5.204 ± .010
	R-R	.362 ± .015	.937 ± .044	1.858 ± .080	3.028 ± .121	4.294 ± .153
8×8	B-P	.336 ± .002	.987 ± .025	2.735 ± .020	6.820 ± .167	10.130 ± .377
w/ Mux	R-R	.300 ± .010	.699 ± .029	1.298 ± .053	2.234 ± .045	3.618 ± .149
Modified	B-P	.2701 ± .0008	.864 ± .004	1.716 ± .008	2.538 ± .019	3.307 ± .038
Delta	R-R	.479 ± .024	1.200 ± .068	2.390 ± .085	3.854 ± .099	5.360 ± .104

equals to 0.5. The 8×8 network with multiplexer and demultiplexer has the lowest packet loss until the *arrival/process* reaches 0.45. Figure 5.10 is very interesting. It shows the expect time delay for these models using the back-pressure algorithm. The 8×8 network with multiplexer and demultiplexer has the fastest time delay increase. At low utilization, the expected delay is very good, only slightly longer than the time delay in model for modified 8×8 delta network. But at 0.2 utilization it soars up. The expected packet time delay is 10.13 unit time at 0.5 utilization. The modified 8×8 delta network maintains the lowest packet time delay from 0.1 to 0.5 utilization.

Figure 5.11 is the packet loss probability of three switches with reject-retransmission algorithm, for non-uniform arrival rates and routing probabilities. Figure 5.12 is the expected packet time delay of these models. At low utilizations, the packet loss probabilities of three designs are very close (about 0.017). However, as utilization (arrival/processing ratio) increases, the three models display different loss probabilities. 8×8 network with mutiplexer and demultiplexer holds the lowest loss probability and modified delta network has the highest – 41% of the packets are lost at the input queues at 0.5 utilization.

Figure 5.12 shows the expected packet time delay of models with reject-retransmission algorithm under non-uniform workload and routing probabilities. Modified delta network displays the longest delay of the three when switch utilization is between 0.1 and 0.5. The model for network with multiplexer and demultiplexer shows the lowest delay at all utilizations. The expected time

Figure 5.9: Packet Loss Probability of Switch Models at Non-uniform Workload, Non-uniform Routing Probability (Back-Pressure)

Figure 5.10: Expected Time Delay of Switch Models at Non-uniform Workload, Non-uniform Routing Probability (Back-Pressure)

needed for packet to propagate through the system increases in basic banyan model maintains the medium of the three designs.

5.4 Discussion

Observing the curves for the three designs under different workloads and routing probabilities, we can know the behavior of these switches in general. The design with multiplexer and demultiplexer performs the best overall using reject-retransmission algorithm. It is especially good in handling the non-uniform workload and routing probabilities. The modified delta network, in contrast, performs the best overall when back-pressure algorithm is used. Under uniform workload and routing probability, it handles particularly well. The design with multiplexer and demultiplexer, probably due to its non-symmetric layout, always shows a sharp increase on the characteristic curve after a certain point is reached.

Figure 5.11: Packet Loss Probability of Switch Models at Non-uniform Workload, Non-uniform Routing Probability (Reject-Retransmission)

Figure 5.12: Expected Time Delay of Switch Models at Non-uniform Workload, Non-uniform Routing Probability (Reject-Retransmission)

CHAPTER 6

CONCLUSION

When constructing a communication network, it is vital to choose a switching technique which is suitable for the application. Packet switching, which acquires and releases the bandwidth as needed, is a more flexible way to utilize the resources. However, some problems such as congestion and long delay required attention. This prompted the development of a switch which makes use of hardware implementation of basic switching and protocol functions, known as "fast packet switching". A class of fast packet switches is based on the design using multi-stage interconnection networks. Banyan networks are the generic name for these multi-stage interconnection networks.

We examined three types of switching fabric based on banyan networks.

1. an 8×8 modified delta network.
2. an 8×8 switch with 4×4 banyan networks, multiplexers and demultiplexers.
3. an 8×8 baseline banyan network.

Two contention resolution algorithms were investigated associating with these fabrics, namely back-pressure and reject. The switch are modeled using stochastic activity networks. Simulations were adopted to obtain the performance variables.

Among the three designs, the one with multiplexer and demultiplexer performs the best overall using reject algorithm. It is especially good in handling the non-uniform workload and routing probabilities. Whereas the modified delta network performs the best overall when back-pressure algorithm is used. Under uniform workload and routing probability, it handles particularly well.

For the future work, it will be interesting to investigate the performance of different sizes of switch network under different workloads. Finding out the congestion points (bottleneck) in the switches is another direction of research for a better understanding of these designs.

BIBLIOGRAPHY

[1] K. E. Batcher, "Sorting networks and their applications," *Proceeding of the Spring Joint Computer Conference, AFIPS Press*, pp. 307–314, 1968.

[2] D. Dias and J. Jump, "Analysis and simulation of buffered delta network," *IEEE Transactions on Computers*, vol. c-30, no. 4, pp. 273–282, April 1981.

[3] D. Dias and J. Jump, "Packet switching interconnection networks for modular systems," *IEEE Computer*, vol. 14, no. 12, pp. 43–53, December 1981.

[4] D. Dias and M. Kumar, "Packet switching in n log n multistage networks," *Proc. GLOBECOM'84, Atlanta, GA*, pp. 43–53, December 1984.

[5] T. Feng, "A survey of interconnection networks," *IEEE Computer*, vol. 14, no. 12, pp. 12–27, December 1981.

[6] L. Goke and G. Lipovski, "Banyan networks for partitioning multiprocessor systems," in *Proceeding of 1st Annual International Symposium of Computer Architecture*, pp. 21–28. International Symposium of Computer Architecture, December 1973.

[7] L. R. Goke and G. J. Lipovski, "Banyan networks for partitioning multiprocessor systems," *Proceeding of 1st Annual Symposium on Computer Architecture*, pp. 21–28, 1973.

[8] A. Huang and S. Knauer, "Starlite: A wideband digital switch," in *Proceeding of 1984 GLOBECOMM*. GLOBECOMM, 1984.

[9] J. Hui and E. Arthurs, "A broadband packet switch for integrated transport," *IEEE Journal on Selected Areas in Communications*, vol. SAC-5, no. 8, pp. 1264–1273, October 1987.

[10] Y. Jenq, "Performance analysis of a packet switch based on single-buffered banyan network," *IEEE Journal on Selected Areas in Communications*, vol. SAC-1, no. 6, pp. 1014–1021, December 1983.

[11] H. Kim and A. Leon-Garcia, "Performance of buffered baynan networks under nonuniform traffic patterns," *IEEE INFOCOM'88 New Orleans*, pp. 344–353, March 1988.

[12] C. Kruskal and M. Snir, "The performance of multistage interconnection networks for multiprocessors," *IEEE Transactions on Computers*, vol. c-32, no. 12, pp. 1091–1098, December 1983.

[13] J. J. Kulzer and W. A. Montgomery, "Statistical switching architectures for future services," *Proceeding of ISS'84*, pp. 1–6, 1984.

[14] M. Kumar and J. Jump, "Performance of unbuffered shuffle-exchange networks," *IEEE Transaction on Computers*, vol. c-35, no. 6, pp. 573–578, June 1986.

[15] Private discussions with Dr. M. Liu in Department of Electrical and Computer Engineering, University of Arizona, 1989–1990.

[16] J. C. McDonald, editor, *Fundamentals of Digital Switching*, Plenum Press, 1983.

[17] J. F. Meyer, "Performability modeling of distributed real-time systems," in *Mathematical Computer Performance and Reliability*, Amsterdam: North-Holland, 1984.

[18] J. F. Meyer, A. Movaghar, and W. H. Sanders, "Stochastic activity networks: structure, behavior, and application," *Proc. International Workshop on Timed Petri Nets Torino,Italy*, pp. 106–115, July 1985.

[19] S. E. Minzer, "Broadband ISDN and asynchronous transfer mode (ATM)," *IEEE Communications Magazine*, pp. 17–24, September 1989.

[20] M. K. Molloy, "Performance analysis using stochastic petri nets," *IEEE Transactions on Computers*, pp. 913–917, September 1982.

[21] A. Movaghar and J. F. Meyer, "Performability modeling stochastic activity networks," *IEEE 1984 Real-Time Symposium, Austin, Texas*, pp. 215–224, December 1984.

[22] P. Newman, "A broad-band packet switch for multi-service communications," *IEEE INFOCOM'88, New Orleans*, pp. 19– 28, March 1988.

[23] P. Newman, "A fast packet switch for the integrated services backbone network," *IEEE Journal on Selected Areas in Communications*, vol. 6, no. 9, pp. 1468–1479, December 1988.

[24] J. Patel, "Performance of processor-memory interconnections for multiprocessors," *IEEE Transactions on Computers*, vol. c-30, no. 10, pp. 771–780, October 1981.

[25] J. H. Patel, "Processor-memory interconnections for multiprocessors," *Proceeding of 6th Annual Symposium on Computer Architecture*, pp. 168–177, April 1979.

[26] W. H. Sanders, "Construction and solution of performability models based on stochastic activity networks," *Computing Research Laboratory Technical Report CRL-TR-9-88, The University of Michigan, Ann Arbor, MI*, August 1988.

[27] W. H. Sanders and J. F. Meyer, "METASAN: A performability evaluation tool based on stochastic activity networks," in *Proc. ACM-IEEE Comp. Soc. 1986 Fall Joint Comp. Conf.*, Dallas, TX, November 1986.

[28] W. Stallings, *Data and Computer Communications*, Macmillan Publishing Co., Inc., 1986.

[29] W. Stallings, *ISDN an Introduction*, Macmillan Publishing Co., Inc., 1989.

[30] J. Turner, "Design of an integrated services packet network," *ACM*, pp. 124–133, 1985.

[31] H. Uematsu and R. Watanabe, "Architecture of a packet switch based on banyan switching network with feedback loop," *IEEE journal on Selected Areas in Communications*, vol. 6, no. 9, pp. 1521–1527, December 1988.

[32] L. Wu, "Mixing traffic in a buffered banyan network," *ACM*, pp. 134–139, 1985.