1-1-1996

A Performance evaluation of several ATM switching architectures

Jeffrey Krieger

Follow this and additional works at: http://scholarworks.rit.edu/theses

Recommended Citation

Accessed from

This Thesis is brought to you for free and open access by the Thesis/Dissertation Collections at RIT Scholar Works. It has been accepted for inclusion in Theses by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.
A Performance Evaluation of Several ATM Switching Architectures

by

Jeffrey L. Krieger

A Thesis Submitted
in
Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Computer Engineering

Approved by:

Principal Advisor
Roy S. Czernikowski, Professor

Committee member
George A. Brown, Professor

Committee member
Charles Kevin Shank, Assistant Professor

Department of Computer Engineering
College of Engineering
Rochester Institute of Technology
Rochester, New York
January 1996
Title: A Performance Evaluation of Several ATM Switching Architectures

I, Jeffrey L. Krieger, hereby grant permission to Wallace Memorial Library to reproduce my thesis in whole or part.

Signature: ________________________________

Date: January 19, 1996
ABSTRACT

The goal of this thesis is to evaluate the performance of three Asynchronous Transfer Mode switching architectures. After examining many different ATM switching architectures in literature, the three architectures chosen for study were the Knockout switch, the Sunshine switch, and the Helical switch. A discrete-time, event driven system simulator, named ProModel, was used to model the switching behavior of these architectures. Each switching architecture was modeled and studied under at least two design configurations. The performance of the three architectures was then investigated under three different traffic types representative of traffic found in B-ISDN: random, constant bit rate, and bursty. Several key performance parameters were measured and compared between the architectures. This thesis also explored the implementation complexities and fault tolerance of the three selected architectures.
# Table of Contents

List of Figures ................................................................. ii
List of Tables ................................................................. iv
Glossary ............................................................................. v
Architecture and Simulation Variables ................................ xiii

1 Introduction ...................................................................... 1
   1.1 Evolution of ATM .......................................................... 2
   1.2 The ATM Protocol ......................................................... 7
   1.3 B-ISDN and ATM ........................................................... 17
   1.4 ATM Switches ............................................................... 21

2 ATM Switch Architectures ............................................... 33
   2.1 Knockout Switch ........................................................... 34
   2.2 Sunshine Switch ........................................................... 47
   2.3 Helical Switch ............................................................... 58
   2.4 Starlite Switch ............................................................... 68
   2.5 Shared Buffer Memory Switch ......................................... 70

3 Architecture Evaluation .................................................. 72
   3.1 Simulator ................................................................ 73
   3.2 Simulation Study Plan ................................................... 74
   3.3 Performance Parameters Studied ..................................... 88
   3.4 Performance Parameters Not Studied .............................. 89
   3.5 Simulation Considerations ............................................. 90

4 Results and Analysis ..................................................... 95
   4.1 Random Traffic Simulation Results ................................. 95
   4.2 CBR Traffic Simulation Results ...................................... 105
   4.3 Bursty Traffic Simulation Results .................................... 114
   4.4 Switch Complexity ....................................................... 127
   4.5 Other Characteristics .................................................. 136

5 Conclusions .................................................................... 139
   5.1 ATM Architectures ....................................................... 139
   5.2 Problems ................................................................... 141
   5.3 Suggestions for Improvement ......................................... 142
   5.4 Future Study .............................................................. 142

6 References ..................................................................... 144

Appendix A Knockout Switch Bursty Model Text ........................ 147
Appendix B Sunshine Switch Bursty Model Text ........................ 157
Appendix C Helical Switch Bursty Model Text ............................ 166
Appendix D Special Thanks to ............................................... 187
**List of Figures**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>The Single Network View</td>
<td>3</td>
</tr>
<tr>
<td>Figure 2</td>
<td>User Access to different sub-networks in ISDN [?, p. 7]</td>
<td>5</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Overview of B-ISDN configuration [?, p. 12]</td>
<td>7</td>
</tr>
<tr>
<td>Figure 4</td>
<td>The ATM Concept [?, p. 274]</td>
<td>8</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Cell Format at UNI and NNI [?, p. 25]</td>
<td>11</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Relationship between Virtual Paths, Virtual Channels, and Physical Paths [?, p. 284]</td>
<td>14</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Routing in an ATM Network</td>
<td>16</td>
</tr>
<tr>
<td>Figure 8</td>
<td>B-ISDN Protocol Reference Model and Layer Functions [?, p. 59]</td>
<td>17</td>
</tr>
<tr>
<td>Figure 9</td>
<td>ATM Switch Components [?, p. 86]</td>
<td>22</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Input Module Functional Blocks [?, p. 98]</td>
<td>23</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Output Module Functional Blocks [?, p. 134]</td>
<td>24</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Shared Memory Architecture [?, p. 146]</td>
<td>28</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Shared Medium Architecture [?, p. 148]</td>
<td>29</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Fully Interconnected Architecture with Output Buffers [?, p. 98]</td>
<td>30</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Fully Interconnected Architecture with Input Buffers [?, p. 94]</td>
<td>30</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Space Division Architecture</td>
<td>31</td>
</tr>
<tr>
<td>Figure 17</td>
<td>2x2 Switching Element [?, p. 103]</td>
<td>32</td>
</tr>
<tr>
<td>Figure 18</td>
<td>Knockout Switch Concept [?]</td>
<td>34</td>
</tr>
<tr>
<td>Figure 19</td>
<td>Knockout Switch Architecture [?]</td>
<td>35</td>
</tr>
<tr>
<td>Figure 20</td>
<td>Bus Interface Unit Block Diagram [?, p. 173]</td>
<td>36</td>
</tr>
<tr>
<td>Figure 21</td>
<td>Construction of KOS Concentrator Unit [?, p. 124]</td>
<td>38</td>
</tr>
<tr>
<td>Figure 22</td>
<td>Operation of the KOS Shifter Unit [?]</td>
<td>40</td>
</tr>
<tr>
<td>Figure 23</td>
<td>Concentrator Cell Loss Probabilities at 90% Traffic Load for Various $L$ and $N$</td>
<td>42</td>
</tr>
<tr>
<td>Figure 24</td>
<td>Discrete-time Markov Chain Model for the Shared Buffer [?]</td>
<td>44</td>
</tr>
<tr>
<td>Figure 25</td>
<td>Shared Buffer Cell Loss Probabilities for Various Traffic Loads and Buffer Sizes</td>
<td>46</td>
</tr>
<tr>
<td>Figure 26</td>
<td>Sunshine Architecture Block Diagram [?]</td>
<td>47</td>
</tr>
<tr>
<td>Figure 27</td>
<td>Binary Tree and Banyan Network Topologies [?, p. 217]</td>
<td>48</td>
</tr>
<tr>
<td>Figure 28</td>
<td>Batcher Sorting Network Layout and Routing Example</td>
<td>50</td>
</tr>
<tr>
<td>Figure 29</td>
<td>The Problem with a Single Batcher Based Concentrator</td>
<td>53</td>
</tr>
<tr>
<td>Figure 30</td>
<td>Concentrator Cell Output Order</td>
<td>54</td>
</tr>
<tr>
<td>Figure 31</td>
<td>Recirculator Cell Loss Probabilities for Various Traffic Loads and one Banyan [?]</td>
<td>57</td>
</tr>
<tr>
<td>Figure 32</td>
<td>Recirculator Cell Loss Probabilities for 100% Load and $k$ Banyan Networks [?]</td>
<td>57</td>
</tr>
<tr>
<td>Figure 33</td>
<td>Helical Architecture Block Diagram [?]</td>
<td>59</td>
</tr>
<tr>
<td>Figure 34</td>
<td>Operation of the Broadcast Unit [?]</td>
<td>60</td>
</tr>
<tr>
<td>Figure 35</td>
<td>Operation of the Helical Concentrator [?]</td>
<td>61</td>
</tr>
<tr>
<td>Figure 36</td>
<td>Importance of the Broadcast Unit [?]</td>
<td>62</td>
</tr>
<tr>
<td>Figure 37</td>
<td>Construction of Helical Concentrator [?]</td>
<td>63</td>
</tr>
<tr>
<td>Figure 38</td>
<td>Routing of Cells through a 4x4 Helical Switch</td>
<td>65</td>
</tr>
<tr>
<td>Figure 39</td>
<td>Starlite Architecture Block Diagram [?, p. 216]</td>
<td>68</td>
</tr>
<tr>
<td>Figure 40</td>
<td>Shared Buffer Memory Architecture [?, p. 146]</td>
<td>70</td>
</tr>
<tr>
<td>Figure 41</td>
<td>ProModel Location and Routing Layout of Knockout Switch Model</td>
<td>75</td>
</tr>
<tr>
<td>Figure 42</td>
<td>ProModel Location and Routing Layout of Helical Switch Model</td>
<td>76</td>
</tr>
<tr>
<td>Figure 43</td>
<td>ProModel Location and Routing Layout of sunshine Switch Model</td>
<td>76</td>
</tr>
<tr>
<td>Figure 44</td>
<td>Illustration of Constant Bit Rate Traffic Implementation</td>
<td>79</td>
</tr>
<tr>
<td>Figure 45</td>
<td>Illustration of Bursty Traffic</td>
<td>80</td>
</tr>
<tr>
<td>Figure 46</td>
<td>CLP of $K_1$, $S_1$, $S_2$, and $H_1$ Buffer 1 Architectures under Random Traffic</td>
<td>96</td>
</tr>
<tr>
<td>Figure 47</td>
<td>CLP of $K_2$, $S_3$, $S_4$, and $H_2$ Buffer 2 Architectures under Random Traffic</td>
<td>97</td>
</tr>
<tr>
<td>Figure 48</td>
<td>CLP of Sunshine Configurations $S_1$, $S_2$, $S_3$, $S_4$, and $S_5$ under Random Traffic</td>
<td>97</td>
</tr>
<tr>
<td>Figure 49</td>
<td>Cell Delay of $K_1$, $S_1$, $S_2$, and $H_1$ Architectures under Random Traffic</td>
<td>98</td>
</tr>
</tbody>
</table>
List of Tables

Table 1 Service Class Characteristics ........................................... 19
Table 2 SONET/SDH Transmission Rates ........................................ 21
Table 3 CLP Comparison of Starlite and Sunshine Architectures ................ 69
Table 4 Simulated Architecture Configurations Described with Parameters Sets ........................................ 85
Table 5 Simulated Traffic Configurations Described with Parameters Sets ........ 86
Table 6 Simulation Scenarios Performed .......................................... 87
Table 7 Model Simulation Times .................................................. 94
Table 8 Statistics on Performed Simulations ..................................... 94
Table 9 Throughput Ranges of Buffer 1 Architectures under Random Traffic ........................................ 103
Table 10 Throughput Ranges of Buffer 2 Architectures under Random Traffic ........................................ 103
Table 11 Throughput Ranges for Buffer 1 Architectures under CBR, Nonrepeated Traffic ........................................ 106
Table 12 Throughput Ranges for Buffer 2 Architectures under CBR, Nonrepeated Traffic ........................................ 106
Table 13 Totals Cells Submitted in CBR, Repeated Traffic for Various Frames Sizes ........................................ 110
Table 14 Cells Recirculated for Various Values of $k$ and $T$ in the Sunshine Architecture under CBR, Repeated Traffic, Frame = 50 cells ........................................ 114
Table 15 Cells Lost at the Recirculator for Various Values of $k$ and $T$ in the Sunshine Architecture under CBR, Repeated Traffic, Frame = 50 cells ........................................ 114
Table 16 Cells Lost at Buffers for Various Values of $k$ and $T$ in the Sunshine Architecture under CBR, Repeated Traffic, Frame = 50 cells ........................................ 114
Table 17 Worst Case Simulated Throughput Ranges for Buffer 1 Architecture Configurations under Bursty Traffic ........................................ 124
Table 18 Worst Case Simulated Throughput Ranges for Buffer 2 Architecture Configurations under Bursty Traffic ........................................ 124
Table 19 Bursty Traffic Equivalent Loads ......................................... 125
Table 20 Effects of Bursty Traffic on Equivalent Load for Constant CLP ........................................ 127
Table 21 Definitions and Basic Building Blocks for Complexity Estimate ........................................ 128
Table 22 Larger Building Blocks for Architecture Complexity Estimates ........................................ 129
Table 23 Complexity of Knockout Switch Components ........................................ 130
Table 24 Knockout Switch Complexity Estimate for Various Configurations in transistors ........................................ 130
Table 25 Complexity of Sunshine Switch Components ........................................ 132
Table 26 Sunshine Switch Complexity Estimate for Various Configurations in transistors ........................................ 132
Table 27 Helical Switch Complexity Estimate for Various Configurations in transistors ........................................ 134
Table 28 Comparison of the Complexity for the 3 Architectures ........................................ 135
This glossary provides a brief list of terms and definitions relevant to topic of this thesis, Asynchronous Transfer Mode. For each entry, a section pointer and page number are provided to the first significant occurrence of the listed term in the main text. Many of the following definitions have been included directly from [7]. At the end of this glossary on page xiii, a separate section containing the Architecture and Simulation Variables is included for quick reference.

**AAL Connection**, section N/A, p. N/A
Association established by the ATM Adaptation Layer (AAL) between two or more next higher layer entities in the B-ISDN protocol reference model.

**Assigned cell**, section N/A, p. N/A
A cell which provides a service to an application using the ATM layer service.

**ATM Adaptation Layer (AAL)**, section 1.3.4, p. 19
A layer in the B-ISDN protocol reference model which is responsible for mapping data between high layer services in the model and the ATM layer.

**Automatic Repeat Request (ARQ)**, section 1.2.2.1, p. 9
A type of error correction protocol in which a receiver requests the retransmission of errored data.

**Asynchronous Time Division Multiplexing**, section 1.2.1, p. 7
A multiplexing technique in which a transmission capability is organized in a priori unassigned time slots. The time slots are assigned to cells upon request of each applications instantaneous real need.

**Asynchronous Transfer Mode (ATM)**, section 1, p. 1
A transfer mode in which the information is organized into cells. It is asynchronous in the sense that the recurrence of cells containing information from an individual user is not necessarily periodic.

**ATM Connection**, section 1.2.2.3, p. 10
As association established by the ATM Layer to support communications between two or more ATM service users (i.e., two or more next-higher layer entities). The communications over an ATM Layer connection may be either bi-directional, two VCCs are used. When unidirectional only one VCC is used.

**ATM Link**, section 1.2.2.6, p. 15
A virtual path link (VPL) or a virtual channel link (VCL).

**ATM Switch**, section 1.4, p. 21
The basic component of the ATM / B-ISDN network which performs multiplexing and routing of ATM cells to transport them to the correct destination. The switch is composed of several components.
ATM Traffic Descriptor, section N/A, p. N/A

A generic list of traffic parameters that can be used to capture the intrinsic traffic characteristics of an ATM connection.
Broadband, section 1.1.5, p. 6
A service or system requiring transmission channels capable of supporting rates greater than the ISDN primary rate.

Bit Error Rate (BER), section 1.2.2.1, p. 9
The rate at which bit errors occur in the physical transmission of a digital signal.

Broadband Integrated Services Digital Network (B-ISDN), section 1.1.5, p. 6
A digital network capable of supporting a wide range of user services regardless of the characteristics of the traffic it generates.

Call Admission Control (CAC), section 1.4, p. 21
A component of the ATM switch which is responsible for processing signalling information and rejecting or granting user requests for an ATM connection to the switch.

Cell, section 1.2.1, p. 7
The basic protocol data unit used by the ATM Layer to transport data over an ATM connection.

Cell Delay, section 1.4.5.1, p. 27
The time a cell remains within a switching architecture.

Cell Delay Variation, section 1.4.5.1, p. 27
A quantification of variability in cell delay for a connection.

Cell Header, section 1.2.2.2, p. 9
The ATM Layer protocol control information stored at the head of the cell.

Cell Error Probability (CEP), section 3.4, p. 90
The ratio of the number of erred cells to the total number of cell submitted to an ATM switch.

Cell Loss Probability (CLP), section 1.2.2.4 & 3.3, p. 12 & 12, 88
The ratio of the number of cells discarded by an ATM switch to the total number of cells submitted to the switch.

Cell Loss Priority (CLP), section 1.2.2.3, p. 11
A one bit field in the ATM cell header which assigns one of two levels of loss priority to a cell. This information is used a selective cell discard algorithm to determine which cell to discard during times of congestion within a switch.

Cell Misinsertion Rate (CMR), section 3.4, p. 90
A cell misinsertion occurs when a cell arrives at a destination for which it was not intended to arrive at, mostly caused by an erred cell.

Cell Switching Fabric (CSF), section 1.4, p. 21
The interconnection fabric of the ATM switch which routes cells from the switch input modules (IMs) to the correct switch output modules (OMs).

Cyclical Redundancy Check (CRC), section 1.2.2.3, p. 10
An error correction protocol which attaches a code generated by division of the data to be protected by a predetermined polynomial. The length of the generate code determines the degree of error protection provided.
Confidence Interval, section 3.5.1, p. 91
An interval determined from multiple sample points of a random variable $x$ in which the true average of the variable is expected to be with a probability of $\alpha$, called the confidence interval level.

Constant Bit Rate (CBR), section 1.1.5, p. 6
A traffic source which generates data at a fixed, periodic rate, like digital telephone data.

Convergence Sublayer (CS), section 1.3.4, p. 19
A sublayer in the ATM Adaption Layer in the B-ISDN protocol reference model which performs functions specific to the different service classes.

Demultiplexing, section N/A, p. N/A
A function performed by a layer entity that identifies and separates SDUs from a single connection to more than one connection.

Errored Cell, section 3.4, p. 90
A cell whose header has errors and has not been modified by the cell header error control verification process. The cell is discarded at the physical layer.

Frequency Division Multiplexing (FDM), section 1.1.3, p. 4
A multiplexing technique in which a transmission capability is organized into different frequencies. Multiple channels are merged together onto the single medium by transmitting each channel at a specific frequency.

General Flow Control (GFC), section 1.2.2.3, p. 10
A four bit field in the ATM cell header which is used to control the flow of data at the user network interface.

Head of Line (HOL), section 2.1.1, p. 34
The position from which the next cell in a cell buffer/queue will be removed and processed.

Header Error Control (HEC), section 1.2.2.3, p. 12
An eight bit field in the ATM cell header which contains a CRC code to perform header error control procedures to protect the sensitive fields contained within the cell header.

Helical Switch (HES), section 2.3, p. 58
One of the three ATM switch architectures evaluated in this thesis.

Idle cell, section 1.3.6, p. 20
A cell that is inserted/extracted by the physical layer in order to adapt the cell flow rate at the boundary between the ATM layer and the physical layer to the available payload capacity of the transmission system used.

Input Module (IM), section 1.4, p. 21
A module in the ATM switch which receives cells from the physical layer of the B-ISDN protocol reference module and preprocesses the cells prior to forwarding them to the CSF.

Integrated Digital Network (IDN), section 1.1.3, p. 4
A name given to the upgrade of the analog public switch telephone network to digital technology, where data of different service types are multiplexed together onto the same physical medium.
**Integrated Services Digital Network (ISDN),** section 1.1.4, p. 5
The evolution of the IDN to included standards for user access to various digital subnetworks by specifying two channel types, Basic channel (two 64Kbps data lines and one 16kbps signalling line) and Primary channel (23 64kpbs data lines and one 64kpbs signalling line).

**International Telecommunications Union (ITU),** section 1, p. 1
An international standard organization responsible for many of the proposals, recommendation, and standardization of protocols for telecommunication technologies, including ATM. It was formerly referred to as CCITT.

**Knockout Switch (KOS),** section 2.1, p. 34
One of the three ATM switch architectures evaluated in this thesis.

**Layer Entity,** section N/A, p. N/A
An active element within a layer.

**Layer Function,** section N/A, p. N/A
An action performed the active layer entities.

**Layer Service,** section 1.3.2, p. 18
A capability of a layer and the layers beneath it that is provided to the upper layer entities at the boundary between that layer and the next higher layer.

**Layer User Data,** section 1.3.2, p. 18
Data transferred between corresponding entities on behalf of the upper layer or layer management entities for which they are provided.

**Markov Chain,** section 2.1.3, p. 43
A special stochastic process modeled by a chain of states which is characterized by the Markov (memoryless) property, where the state of the chain is dependent only upon the previous state of the chain.

**Multistage Interconnection Network (MIN),** section 1.4.5.2.4, p. 31
A type of CSF constructed from multiple stages of a basic switching element.

**Multiplexing,** section N/A, p. N/A
A function with a layer that interleaves the information from multiple connections into one connection.

**Network to Network Interface (NNI),** section 1.3.4, p. 19
The interface between two network nodes.

**Narrowband Integrated Service Digital Network (N-ISDN),** section N/A, p. N/A
The name given to original definition and specifications of ISDN with the introduction of B-ISDN.

**Operation and Maintenance (OAM) cell,** section 1.4.4, p. 25
A cell that contains ATM Layer Management information. It does not form part of the higher layer information transfer.

**Output Module (OM),** section 1.4, p. 21
A module in the ATM switch which prepares cells received from the CSF for physical transmission on the physical layer of the B-ISDN protocol reference model.
Peak Cell Rate (PCR), section 1.2.2.5, p. 13
The peak cell rate traffic parameter which specifies the upper bound on the traffic bandwidth submitted on an ATM connection.

Packet Error Rate (PER), section 1.2.2.1, p. 9
See Cell Error Probability (CEP).

Packet Insertion Rate (PIR), section 1.2.2.1, p. 9
See cell misinsertion rate (CMR).

Packet Loss Rate (PLR), section N/A, p. N/A
See cell loss probability (CLP).

Packet Switched Data Network (PSDN), section 1.1.1, p. 2
A data network in which information is carried in discrete packets which possess a source and destination address which directs the packets routing at each node in the switched network.

Payload Type (PT), section 1.2.2.3, p. 11
An 3-bit field in the ATM cell header which identifies the type of information carry within the payload section of the cell. This data may be either user data or network management data to which virtual channel the cell belongs. It may also indicate network status.

Plain Old Telephone Service (POTS), section 1.1.1, p. 2
Classical two way voice communications over the public switched telephone network.

Protocol, section N/A, p. N/A
A set of rules and formats (semantics and syntactics) that determines the communication behavior of layer entities in the performance of layer functions.

Protocol Data Unit (PDU), section 1.3.4, p. 19
A unit of data specified in a layer protocol and consisting of protocol control information and layer user data.

Public Switched Telephone Network (PSTN), section 1.1.1, p. 2
A public circuit switched network used POTS.

Quality of Service (QOS) Class, section 1.2.2.4, p. 12
A Quality of Service class, which is negotiated during the call setup phase of an ATM connection, specifies the required performance level through the use of traffic description parameters which must provided by the ATM network for the duration of the connection.

Random Access Memory (RAM), section 2.4.3, p. 69
A device used to store digital values and allow random access to that stored information.

Running Adder Network (RAN), section 2.3.2.4, p. 63
A network of adders used to implement the concentrator unit of the Helical switch architecture.

Segmentation and Reassembly Sublayer (SAR), section 1.3.4, p. 19
A sublayer in the ATM Adaption Layer in the B-ISDN protocol reference model which is responsible for mapping variable length protocol data units (PDUs) passed to it from the convergence sublayer (CS) functions into fixed length ATM cell, and vice-versa.
Semi-Permanent Connection, section 1.2.2.6, p. 15
A connection established via a service order or via a network management.

Sustainable Cell Rate (SCR), section 1.2.2.5, p. 13
A traffic parameter which describes the upper bound on the realized average cell rate of an ATM connection. The value of the SCR parameter will be below the peak cell rate (PCR).

Service Class, section 1.3.3, p. 18
A group of user services like telephony, data transfer, and video distribution which exhibit the same source traffic descriptors. Four service classes have been defined to date by ITU-T.

Service Data Unit, section N/A, p. N/A
A unit of interface information whose identity is preserved from one end of a layer connection to the other.

Simulation Scenario, section 3.2.4, p. 85
A specific set of values chosen for the variable parameters associated with the architectural models and the different traffic types.

Simulation Replication, section 3.2.4, p. 85
A single execution of a simulation model for a given simulation scenario.

Source Traffic Descriptor, section 1.3.3, p. 18
A set of traffic parameters belonging to the ATM Traffic Descriptor used during the connection setup to capture the intrinsic traffic characteristics of the connection requested by the source.

Stochastic Process, section 2.1.3, p. 43
A collection of random variables which are defined upon a sample space and change over time.

Sunshine Switch (SUN), section 2.2, p. 47
One of the three ATM switch architectures evaluated in this thesis.

Synchronous Digital Hierarchy (SDH), section 1.3.6, p. 20
A particular specification in the physical layer of the B-ISDN protocol reference model which defines a family of data rates and frame formats for interfaces used in fiber-optic transmission systems.

Synchronous Optical Network (SONET), section 1.3.6, p. 20
A particular specification in the physical layer of the B-ISDN protocol reference model which defines a family of data rates and frame formats for interfaces used in fiber-optic transmission systems.

Synchronous Transfer Mode (STM), section 1.2.3, p. 16
A transfer mode which in which information is organized into time slots within a transmission frame. A data unit's connection identification is identified by its temporal position in the frame. (see TDM)

Switch Priority (SP), section 2.2.2.2, p. 49
A priority field added to the cell by the Sunshine switch architecture to prevent the out of sequence problem.

System Management Module (SMM), section 1.4, p. 21
A module in the ATM switch which performs management and traffic control functions.
Time Division Multiplexing (TDM), section 1.2.1, p. 7
A multiplexing technique in which the physical transmission bandwidth is segmented into time slots grouped into a frame. Connections over a TDM system are assigned a specific time slot position within the framed structure and thus is provided a fixed bandwidth.

Traffic Parameter, section 1.3.3, p. 18
A parameter for specifying a particular traffic aspect of a connection.

True average, section 3.5.1, p. 92
The average of a random variable x when an infinite number of sample points are used.

Unassigned Cell, section 2.1.2.1, p. 36
A cell identified by a standardized virtual path identifier (VPI) and virtual channel identifier (VCI) value, which has been generated and does not carry information from an application using the ATM Layer service.

User to Network Interface (UNI), section 1.3.4, p. 19
The interface between an user and a network node.

Valid Cell, section N/A, p. N/A
A cell whose header has no errors or has been modified by the cell header error control verification process.

Variable Bit Rate (VBR), section 1.1.5, p. 6
A traffic source which generates data at a fixed, periodic rate, like digital telephone data.

Virtual Channel (VC), section 1.2.2.6, p. 14
A communication channel that provides for the sequential, unidirectional transport of ATM cells

Virtual Channel Connection (VCC), section 1.2.2.6, p. 15
A concatenation of VCLs that extends between the points where the ATM service users access the ATM layer. The points at which the ATM cell payload is passed to or received from the users of the ATM Layer for processing signify the endpoints of a VCC. VCCs are unidirectional.

Virtual Channel Link (VCL), section 1.2.2.6, p. 15
A means of unidirectional transport of ATM cells between the point where a VCI value is assigned and the point where that value is translated or removed.

Virtual Channel Switch, section N/A, p. N/A
A network element that connects VCLs. It is terminates VPCs and translates VCI values. It is directed by the Control Plane in the B-ISDN protocol reference model.

Virtual Channel Identifier (VCI), section 1.2.2.6, p. 14
An field in the cell header which identifies to which virtual channel the cell belongs.

Virtual Path (VP), section 1.2.2.6, p. 15
An unidirectional logical association or bundles of virtual channels (VCs).

Virtual Path Connection (VPC), section 1.2.2.6, p. 15
A concatenation of virtual path links (VPLs) between virtual path terminators (VPTs). VPCs are unidirectional.
Virtual Path Link (VPL), section 1.2.2.6, p. 15
A means of unidirectional transport of ATM cells between the point where a VPI value is assigned and the point where that value is translated or removed.

Virtual Path Switch, section N/A, p. N/A
A network element that connects VPLs. It translates VPI (not VCI) values and is controlled by the Control Plane in the B-ISDN protocol reference model. It relays the cells of the virtual path (VP).

Virtual Path Identifier (VPI), section 1.2.2.6, p. 15
A field in the cell header which identifies to which virtual path a cell belongs.

Virtual Path Terminator (VPT), section N/A, p. N/A
A system that unbundles the virtual channels (VCs) of a virtual path (VP) for independent processing of each virtual channel (VC).

Videophony, section 1, p. 1
A telephone like service which also provides a video component so that end users can see and converse with each other via a video screen.
Architecture and Simulation Variables

This section of the glossary provides a brief list of variables used to describe the different parameters associated with the various ATM architectures and the simulation traffic types.

\( B \), sections 2.1.2.5, 2.3.3, p. 40, 65
In the Knockout and Sunshine architectures, the amount of cell buffers present at each output port. In the Helical architecture, the depth of cell buffers at each stage of the architecture.

\( F \), section 3.2.2.2, p. 78
In CBR traffic, cells arrival at the input ports in a frame structure. The number of cells in a frame is described is described by \( F \). Exactly \( F \) cells arrive at an input port during a frame cycle, and exactly \( F \) cells are destined to output port during a frame cycle.

\( k \), section 2.2.2.1, p. 48
In the Sunshine architecture, the number of parallel Banyan routing networks.

\( I \), section 3.2.2.3, p. 80
In bursty traffic, a burst of cells arriving at an input port are described by two parameters. The parameter \( I \) specifies the mean length of the burst in cells.

\( L \), section 2.1.2.3, p. 37
In the Knockout switch architecture, the number of output lines from the concentrator units.

\( N \), section 1.4.5, p. 25
The number of input/output ports possessed by an ATM switch.

\( p \), section 3.2.2.1, p. 77
In random traffic, cells arrive at each of the input ports of an architecture independently of each other with a probability \( p \), referred to as the input load. When \( p = 0 \) at an input port, the port receives no active cells. When \( p = 1 \), the input port receives a cell in each time slot.

\( t \), section 3.2.2.3, p. 80
In bursty traffic, a burst of cells arriving at an input port are described by two parameters. The parameter \( t \) specifies mean interarrival time, the time between the end of one burst on an input port and the arrival of the next.

\( T \), section 2.2.2.7, p. 55
In the Sunshine architecture, the number of recirculator lines.
1 Introduction

The evolution of telecommunications has progressed over more than one century, with dramatic progress occurring in the last two decades. Part of this increased development is the result of the self-perpetuating need for more information and data. As society progresses into the future, the demand for information exchange will grow. Today, that demand for information is embodied by many telecommunication services, including telephony, video conferencing, videophony, High Definition Television (HDTV), high speed data transfer, and home entertainment on demand to name just a few. Unquestionably, new services and demands will emerge in the future which no one has even conceived of. Currently, many independent communication networks and protocols designed for transporting a specific service coexist in the world, however, no single communication system exists to fulfill all these services simultaneously. This situation creates numerous problems when planning for the introduction of new services. Either a new network must be designed and installed, or a method must be devised for transporting the new service over an existing network for which the network was not designed. Even though progress has been made upgrading current networks to digital transmission technology to reach new levels of performance, like the Public Switched Telephone Network (PSTN), the fundamental problem just described still remains.

Only seven years ago, the International Telecommunication Union (ITU) announced the standards for a new transport protocol referred to as Asynchronous Transfer Mode (ATM) to move telecommunications beyond this dilemma. Other solutions exists, but the flexibility and simplicity of ATM have made it widely accepted as the protocol for any future broadband network. ATM was designed to be a service-independent transport technique used to provide fast and efficient transmission of data from all forms of services over the Broadband Integrated Services Digital Network (B-ISDN), regardless of the type of data or the amount of bandwidth (within limits) required by the service. An examination of the characteristics of the networks in today's telecommunications networks will provide a better explanation of what ATM is and how exactly it fits into the telecommunications puzzle.

The first several sections of this chapter introduce the current state of the telecommunication networking infrastructure and how it has evolved, what future demands and services it will be required to transport and manage, and how this evolution has resulted in the development of the ATM. The following section will describe the relationship between B-ISDN and ATM and the
different components and concepts defined in their protocols. In the final section, one component in particular, the ATM switching fabric, will be explored since is the main topic of study in this thesis.
1.1 Evolution of ATM

The evolution of telecommunications towards ATM begins by examining the characteristics of the telecommunication networks currently available.

1.1.1 Public Networks

Today, there exist many public networks for communications. The common characteristic between all of them is specialization. In general, each network is designed for a single telecommunications service. Therefore, there exists at least one network, possibly more, for each service. Consider the following networks [1, p. 15].

The Public Switched Telephone Network (PSTN) is used to transport Plain Old Telephone Service (POTS). For over half a century, people have enjoyed two way voice communications over this network.

Computer data is primarily transported over public Packet Switched Data Networks (PSDN) using the X.25 protocol. In industry, this data travels over Local Area Networks (LANs) or Ethernet.

Television is broadcast over three media: radio waves, coaxial tree networks, and mostly recently, satellite based Direct TV networks.

Each of these networks shares three characteristics. First, since each network was specifically designed for a particular telecommunications service, they are called service-dependent networks [1, p. 16]. It becomes a monumental task to transport other services or to introduce new services over these existing networks. For instance, sending computer data over the PSTN can only be accomplished by modulating and then demodulating the digital signal through the use of a modem. This results in many independent networks around the world, each of which must be designed, installed, and maintained separately.

Second, each network lacks the ability to easily adjust to changing service requirements [1, p. 17]. For example, great progress has been achieved in voice and video compression algorithms. Algorithms such as Adaptive Differential Pulse Coded Modulation (ADPCM) and Linear Predictive Coding (LPC) have reduced the bandwidth requirements of a typical 64 kbps voice channel to 32 kbps or lower. When these algorithms are applied to existing networks, problems occur. For instance, the current Integrated Digital Network (IDN), the digital upgrade of PSTN which provides digital transmission of voice and data in interoffice facilities and between central offices, cannot
utilize these advances to deliver higher levels of throughput. IDN is based on Time Division Multiplexing (TDM) which multiplexes 23, 64 kbps voice channels plus one data/signalling channel together into a 1.544 Mbps T1 line. Each connection through IDN is assigned a channel of 64 kbps. By using any of these encoding techniques, the voice data transmitted could be reduced to at least 32 kbps. Since two connections cannot be joined on one IDN channel, however, half of the channel bandwidth would not be used. The situation is common in other networks. When such advances in encoding technologies or advances in physical technologies like CMOS and BiCMOS reduce or alter the bit rate, current transmission systems are inflexible and fail to utilize these advances to realize performance gains because they have no inherent ability for adapting to the lower bit rates without external data rate adaptation.

The last commonality of these networks is their inefficiency. This inefficiency has two aspects. First, because each network was designed to carry a specific service, it must be designed to manage the worst case traffic load. When the traffic load falls below the peak load, a portion of the network bandwidth sits idle. Second, because all these networks are service-dependent, it is impossible for networks of different or of even the same services to share resources which are unused. Resources available to one network cannot be utilized by others.

1.1.2 A Single, Universal Network

After considering the problems of service dependence, inflexibility to accommodate new technologies, and inefficient use of resources exhibited by today's networks, the natural evolution of multiple, service-dependent networks is to merge them into a single, service-independent network. A single broadband network would avoid the problems described previously, and it would realize several advantages. First, designing, installing, and maintaining one network would realize significant economic
savings due to economy of scales as opposed to performing the same activities for numerous, dissimilar networks. Second, a single network would have to have the capability of supporting many services of different data type and rates. Therefore, it could provide efficient resource sharing between services; bandwidth not used by one service could be reallocated for use by another. Lastly, since a single broadband network would have the ability of transporting different data rate services, it would be more capable of adapting to advances in compression algorithms and VLSI design which could alter the data rates of established services. Such advances would simply reduce the network bandwidth consumed by the services for which the advances were intended. Finally, since a single broadband network would be designed to accommodate many types of services, it would be flexible enough to adapt to changing service requirements or to the addition of new, undefined services. Such a network might resemble the one in figure 1.

1.1.3 IDN

The concept of a single network for telecommunication services is not new. AT&T's first president envisioned "one policy, one system, universal service." [2, p. 6] Until recently, though, technology simply did not exist to implement such a network. The first technology advance towards the concept of a single, universal network was the development of solid state electronics in the 1950s which made digital transmission viable. Digital transmission has certain advantages over analog transmission - it is less susceptible to noise, easier to multiplex, easier to regenerate, and easier to add signalling to [2, p. 5]. Consequently, the telecommunications companies began incorporating digital switching and transmission facilities into the PSTN with the unveiling of AT&T's No. 4 Electronic Switching System (ESS) in 1976. The ESS allowed direct coupling of digital transmission lines to switches for the first time since both were now digital. This eliminated the need for expensive analog/digital converters previously required to use digital transmission lines with analog switches. Furthermore, since digital switches use time division multiplexing as opposed to analog's Frequency Division Multiplexing (FDM) to merge multiple connections onto the same line, the possibility of transmitting any type of digital data besides digital voice data became possible. With digital technology replacing analog in the PSTN, the network became referred to as Integrated Digital Network (IDN). Thus, IDN moves networking to a higher level of integration based on the integration of different types of data traffic onto the same physical medium through multiplexing.

1.1.4 ISDN
With digital transmission technologies improving and networks like PSTN being upgraded to use the digital technology, more services like facsimile, teleconferencing, and videophony will be desired over the available networks. As IDN stands today, it is still a circuit switched network primarily used for digitized voice transmission. To accommodate the new data services of the future, IDN will be expanded by the addition of packet-switched subnetworks. Other subnetworks may be added to bring even more services to the consumer through IDN. Accessing multiple subnetworks conveniently, however, introduces many new problems. Recently, though, this problem was tackled when the next step in network evolution, integration at the user access level, was taken. International standards were adopted for the Integrated Services Digital Network (ISDN) in 1984 and 1988 by ITU-T. ISDN has several fundamental characteristics [2, p. 6].

- End-to-end digital connectivity
- Voice and nonvoice services support
- A limited set of standard User-Network Interfaces (UNI)

Basically, ISDN integrates user access to services available on different digital subnetworks. ISDN specifies several channels for this purpose. The primary channel, called the B-channel, is a 64 kbps digital line for user information, be it voice or data. This channel specification is based on sampling a 4 kHz voice at 8000 samples per second with 8-bits per sample. Other channels rates are also provided, including 16 or 64 kbps D channel for signalling or data communications over a packet switched subnetwork, 384 kbps H0 channel, 1.536 Mbps H11 channel, and the 1.920 Mbps H12 channel.

Furthermore, ISDN extends these digital services to the end users with two different User-Network Interfaces (UNI). The first interface, referred to as the basic rate, consists of 2 B-channels and 1 16 kbps D-channel (called "2B+D16") and totals 144 kbps of bandwidth. The other interface, called the primary rate, is composed of 23 B-channels and one 64 kbps D-channel, otherwise known as "23B+D64". Users can transmit a combination of data belonging to different services into these digital channels up to their capacity, and within the ISDN network, the data will be sorted and routed to the appropriate service-specific subnetwork, as seen in figure 2 [3][4, p. 3].
The specification of ISDN brings the world closer to a single broadband network by specifying common user interfaces to digital networks. From the users perspective, there exists only one connection to a network which seamlessly transports many different services.

1.1.5 Broadband - ISDN

Even as ISDN is still being deployed in the markets of Japan, United States, France, and other nations, the evolution of networking has taken another large step forward. Several factors have influenced this step. First, many new services are on the horizon which ISDN currently can not support because of bandwidth limitations. Broadband services like video may require data rates up to 150 Mbps [1, p. 33]. Current ISDN connections do not support these broadband data rates. Multiple basic and primary channels could be combined to achieve the higher date rates, a multi-rate circuit switching approach. Instead of a connection being assigned only one channel, it could allocate up to \( N \) channels [1, p. 52]. So far this option has only been reserved for the videophony service. This solution is also daunted by problems. Data routed over different channels may be subjected to different delay times when switched, causing desynchronization at the receiving terminal. Another solution would be to extend the number of ISDN basic rates offered, however, this significantly increases the complexity of the involved transmission systems, especially where multiplexing and demultiplexing are concerned. With advances in optical transmission systems and research in packet switching, and realizing the need to support future broadband services, a new solution was developed under ITU-T guidance, Broadband - ISDN.

**Figure 2** User Access to different sub-networks in ISDN [2, p. 7]
As envisioned by ITU-T, B-ISDN is a broadband network capable of supporting all of today's narrowband services like voice and data as well as future broadband services like video and interactive multimedia. Additionally, B-ISDN will have to support services of varying traffic characteristics, including Constant Bit Rate (CBR) as well as Variable Bit Rate (VBR), distributed or interactive, and connection oriented or connectionless. The B-ISDN network will also have to support these services by providing virtual connections which may uni or bidirectional, point to point, or point to multipoint connections. For B-ISDN to support all of these services, it will have to integrate all the traffic at the switching level on a single subnetwork [2, p. 18]. Since this subnetwork will be service independent, B-ISDN will realize most of the advantages of a single network integrated at the transmission, user access, and traffic levels as mentioned earlier.

The creation of B-ISDN represents the current stage of evolution in telecommunications networks, and here is where ATM makes it debut. After much consideration and study, ITU-T adopted ATM to be the switching and multiplexing method for the service-independent subnetwork which forms the forms foundation for B-ISDN, as seen in figure 3.

![Figure 3: Overview of B-ISDN configuration](image)

**Figure 3** Overview of B-ISDN configuration [2, p. 12]

### 1.2 The ATM Protocol

The development of ATM was accomplished by the effort put forth by many different groups, including Bell Core, the ATM Forum, ITU, and ANSI. The results of this development will be described in the next two sections. The following subsections will describe the basic definitions and characteristics of ATM, while section 1.3 will illustrate how ATM is incorporated into the B-ISDN model in order to achieve the single network vision.

#### 1.2.1 Definition
Asynchronous Transfer Mode is a service-independent transport technique used to transport streams of fixed-length data packets, called cells, over the B-ISDN using switching and asynchronous time-division multiplexing [1][2][5]. The asynchronous in ATM does not refer to the physical method of cell transmission, which is in fact synchronous in B-ISDN over the SDH/SONET. Instead, it refers to the way in which bandwidth is allocated among users. The bandwidth is divided into time slots much like in time division multiplexing, also referred to as Synchronous Transfer Mode (STM), used in systems like IDN. Unlike TDM where the position of a time-slot in the frame defines which connection the time-slot belongs to, cells belonging to the same connection in ATM do not have to fill predetermined temporal positions within the transmission frame. A connection in ATM, thus, is not represented by a temporal position in the data stream, but instead by an explicit destination header, much like a destination address in packet switched networks. Since the data from a user's input stream is not fixed into a temporal position, the user may acquire more bandwidth as needed, asynchronously, if the bandwidth is available. The asynchronous allocation of time-slots to a user connection allows ATM to transport the wide spectrum of services supported by B-ISDN, which can be seen in figure 4. The data streams from the range of services may either constant, variable, or bursty in nature. To accommodate all these data types, ATM segments the user data into cells. The cells from the various sources are then mixed together and sent to an ATM switch which multiplexes the cells together into a single cell stream. Cells in the stream then contend for vacant time-slots at the output of the switch, waiting for transmission.

Figure 4 The ATM Concept [5, p. 274]
1.2.2 Concepts

Many of the features and qualities of ATM make it both similar yet different from other transfer modes. Overall, ATM most closely resembles the packet switching transport technique, yet fundamental differences regarding assumptions about the physical transmission of cells in the network and the use of asynchronous multiplexing clearly distinguish ATM concepts from those in packet switching. This section describes the major concepts associated with ATM, like the formatting and transfer of information, the methods of cell routing, and more.

1.2.2.1 Link to Link Functionality

Many of the assumptions about the physical transmission systems used to transport cells in the ATM subnetwork have influenced its specification and design. Ideally, the physical networking components should be designed to transfer data from source to destination error free. Such an idea is not possible in the real world at any cost; noise always finds a way to corrupt the transmitted signal, and the electronic processing components in transmission systems occasionally err due to inherent material flaws and fluctuating physical conditions. The combination of these effects results in a network which has a certain probability of erred data, measured in either Bit Error Rate (BER) or Packet Error Rate (PER). In the recent past, physical transmission systems were of much lower quality than today, and a BER of $10^{-6}$ was considered excellent. To make the network appear as a transparent vehicle of data transport to the user despite the relatively high occurrence of erred data, complex error control was woven into the design of the network at every link, as in the X.25 packet switched network. Error control in X.25 consists of CRC generation and checking, bit stuffing, frame boundary checking, flow control, and Automatic Repeat Request (ARQ). The presence of this error control at every link contributes to each link's complexity and processing requirements and reduces the amount of time which could be focused on routing data. Overall, this reduces the maximum theoretical throughput of the network. Other types of packet switching networks, like frame switching and frame relay improve throughput performance over X.25 by removing the burden of functions like ARQ and flow control from each link and moving them to only the links at the source and destination of a connection.

Because of the quality and reliability of modern digital transmission systems, specifically the optical systems like SONET/SDH, ATM was designed based on the assumption that these facilities have
excellent BER (on the order of $10^{-10}$ or lower) [1, p. 59]. Many of the functions commonly performed link to link in older packet switched networks could now be moved to the fringes of the network where the user interfaces the network, referred to as the User-Network Interface (UNI). Based on this assumption, ATM provides no error correction for data on a per link basis, and data is transparently transported through the network. ATM links only protect the header of the ATM cell which contains the routing information for the data in the cell. This is done to help reduce the Packet Insertion Rate (PIR), the rate at which cells arrive at the wrong destination due to errors in the routing information bits in the cell header. Secondly, no flow control exists between ATM links. It is assumed that flow control will be performed at the UNI to prevent traffic congestion within the ATM network. Lastly, many of the other functions from traditional packet switching have been also moved to the UNI like bit stuffing and ARQ.

1.2.2.2 Functionality of Cell Header

As mentioned in the previous section, packet switched networks like X.25 incorporate complex error control and flow control procedures into the network links to compensate for poor transmission systems. Consequently, this increases the processing time at the links. Much of the information necessary for the execution of these functions is contained in the header of the packet. In the ATM protocol, header functionality is reduced [1, p. 60]. The main function of the header is to identify cells belonging to the same connection and to route them accordingly. Because the header processing at an ATM link is reduced, each cell can be processed much faster, improving the throughput of the network. Also, because the processing is significantly reduced, the few functions needed can be implemented in the switching hardware as opposed to implementation in software via a switching processor to achieve even higher throughput rates and smaller delays. The header in ATM is also responsible for a few minor maintenance functions discussed in section 1.4.4.

1.2.2.3 Information Transport

The transferring of information in ATM is very similar to that of packet switching. Data is transferred in cells (packets) which have both a header and data payload. To ensure proper cell sequencing, minimal transfer delay, and tolerable cell loss rates for real time services like voice communications, ATM is connection-orientated. Because of this orientation, data transfer must be preceded by a connection setup phase. This connection is termed virtual because no physical connection really exists between the source and destination locations. After a virtual connection setup phase, cells
in the same virtual connection traverse the same path through the network, but not necessarily the same path through a specific ATM switch, depending on the type of architecture used. This guarantees that cells of a given connection maintain their sequence. Cells from multiple connections are asynchronously time division multiplexed together through the ATM switching network. The cells transported through the network are 53 bytes long. Five bytes are reserved for the header, and the other forty eight bytes contain the data to be transported. From a packet switched network perspective, 53 bytes is a very short packet which incurs a relatively higher overhead of about 10% with a 5 byte header. In general, a cell length in the thousands of bytes would be more desirable. Because a significant portion of the data over ATM will be voice traffic, the length of the cell must be relatively short to reduce packetization delays, the time to fill or unload data stored in the cell's information field. Longer cells would increase this delay and degrade the quality of voice traffic [1, p. 80]. To accommodate both voice and data services, ATM compromises with a 53 byte cell. The definition of the cell header varies slightly depending on where the cell is in the network, either at the UNI or an NNI. The cell headers are defined as seen in figure 5.

The fields of the cell are easily explained [4, p. 90]. The information field, commonly called the cell payload, contains any data to be transported. Regardless of what the data is, it will be transported transparently through the ATM network. Because of this, many services may include a small Cyclical Redundancy Check (CRC) code in the information field to provide the necessary error detection and correction on the information field when the cell reaches its destination. This function is typically performed by the AAL, described in section 1.3.4.

The first field of the cell header is the General Flow Control (GFC) field which is only present at the UNI. It will be used to implement both controlled and uncontrolled transmission protocols. It also provides a mechanism to implement a point to multipoint protocol for allowing multiple users (terminals) to be connected to the same UNI link, similar to the configurations found in LANs. Currently, the GFC is not implemented in standards and has no affect on the data rates of devices connected to the B-ISDN network [4, p. 92].

Skipping the VPI and VCI fields temporarily, the next field of the header is the Payload Type (PT) Field which is three bits in length. This is used to distinguish cells carrying user data from those which contain network information like signalling and Operation and Maintenance (OAM) cells. Besides specifying payload type, it can also be used by the network to signal that traffic congestion
In ATM, a small statistical probability exists that a cell may be lost in transit. The rate at which cells are lost is designed to be extremely low to prevent severe consequences at the destination. This loss may be due to a temporary traffic congestion or transmission problems. In the case where cells are lost due to traffic congestion, a one bit field referred to as the Cell Loss Priority (CLP) field is provided after the PT field [4, p. 95]. This field assigns one of two levels of loss priority to a cell. This information is used by a selective cell discard algorithm to determine which cell to discard during congestion periods. When CLP=0, the loss priority is high; when CLP=1, the loss priority is low. In case of congestion within the network, a cell with a low cell loss priority will be discarded before a cell with a high cell loss priority. This field can be set by the user on a connection basis or a cell to cell basis. VBR traffic sources like encoded video will benefit the most from the CLP field. If a cell carrying crucial information such as frame synchronization and encoding information was not protected with a higher cell loss priority and was lost during transmit, many data cells before or after it in the same connection would become meaningless without that control information. Thus, the CLP can be used to ensure the routing of critical cells over less important cells.

Finally, an eight bit Header Error Control (HEC) field completes the definition of the cell header. This is used to perform a CRC on the bits contained in the header. Since a total of 32 bits are used in the header and are checked against an 8 bit CRC code, the HEC field is capable of single bit
error correction / multiple bit error detection [2, p. 29]. As will be explained in the next section, this field is updated at each switch in the ATM switching network.

1.2.2.4 Quality of Service

Another fundamental concept and concern in ATM is Quality of Service (QOS). When connections are established in ATM, the network examines the resources available within the network and compares this against those required by the subscriber requesting the connection. The connection is granted only if enough resources are currently available to guarantee the level of performance required by all existing connections plus the level of performance required by the type of traffic the subscriber will be sending in the new connection. If enough resources are available, the connection is granted, but the resources necessary to support the new connection are not specifically allocated to the new connection itself. Connection admission and resource allocation in ATM are thus viewed from an aggregate basis. Once the connection is established, traffic over the connection can, however, experience two major type of impairments, cell delay and cell loss. QOS is used to described several ATM performance parameters including cell loss probability, cell misinsertion rate, cell delay, and cell delay variation which may be experienced by the transmitted traffic. From the user's perspective at the UNI, the network must maintain a certain level of QOS for the duration of the connection. Thus, when establishing a virtual connection, a traffic contract is negotiated between the user and the network which establishes the desired QOS. Both the user and the network must adhere to the negotiated QOS. It naturally follows that as more of the available network bandwidth (resources) is consumed, QOS will decrease as cell losses and cell delays increase due to congestion. This decrease in QOS, however, will effect all of the connections which share the same resources in the network, regardless of how much or how little bandwidth any particular connection is using.

QOS is classified into two broad categories, specified and unspecified. The specified QOS class specifies the set of performance parameters to be used for an ATM connection and the value of each parameter. If the QOS includes a cell loss probability parameter, it must include two values. One vade is for the case where a cell's CLP bit is one and the other value is for the case where the CLP bit is zero. So far, ITU-T has identified and defined fours sets of specified QOS classes for classifying the services that B-ISDN, and thus ATM, must support [2, p. 59].

Class A - connection oriented, constant bit rate services with time relations (like voice)
Class B - connection oriented, variable bit rate services with time relations (like video)
Class C - connection oriented, variable bit rate services with no time relations (like data)
Class D - connectionless services like data from LANs

More classes may be defined in the future. If such classes were not defined to distinguish different
services and traffic types, then the strictest QOS would have to be enforced, implying that the
network could not be fully utilized. The other level of QOS is the unspecified QOS class, which has
no performance parameters associated with it [7, p. 319]. The network is at liberty to decide what
degree of performance can and will be guaranteed to connections requesting this QOS class. The
unspecified QOS class can be used in general for services requiring "best effort" service.

1.2.2.5 Statistical Multiplexing

In STM, bandwidth must be allocated to Variable Bit Rate (VBR) traffic sources based on its Peak
Cell Rate (PCR) throughput parameter. By using this upper bound, STM can support the transport
of VBR traffic, but on average, the traffic source will not be able to fill the bandwidth of the allocated
connection to its limit. On the other hand, VBR sources can also be characterized by the
Sustainable Cell Rate (SCR) throughput parameter, which attempts to describe the actual, average
cell rate of the VBR source. This will always be below the PCR (in Constant Bit Rate (CBR) traffic
sources, PCR = SCR). Since ATM uses asynchronous time division multiplexing, it can multiplex
together VBR sources whose total PCR is greater the physical bandwidth capacity of the
transmission line, but only as long as the sum of the SCRs is less than that line capacity. This is
referred to as statistical multiplexing [1, p. 237]. If the number of VBR sources is large and relatively
independent of each other, then only a small probability exists that the instantaneous data rate of
all the sources will exceed the physical link's capacity. The major benefit of this technique is that
more efficient sharing of resources than can be achieved by other techniques like STM.

It should be noted that an inherent danger exists when statistical multiplexing. VBR traffic sources
tend to bursty in nature. It is possible that multiple VBR sources sharing the same switching
resources could bursty simultaneously, overwhelming the buffers within the switching network. Not
only will cells be lost if buffers overflow, but cell delays due to full buffers will be experience. This
will not only affect the connections which caused the congestion, but could possibly affect all the
other connections sharing the same switching resources and links. Thus, achieving optimal
statistical multiplexing conflicts with the notion of QOS discussed earlier. For this reason, ITU-T has
only specified PCR as a parameter for negotiating an ATM connections while parameters for
determining how to acceptance VBR connection requests based on SCR are studied further to ensure reliable and efficient performance.

1.2.2.6  Routing

Routing in ATM is connection-oriented for reasons explained earlier. Because of the 'connection-oriented' characteristic, ATM does not need a source address, destination address, or sequence number like in packet switched networks. Instead, routing through an ATM network is determined by the cell identifier which uniquely defines a virtual connection. A virtual connection is established as part of the call setup procedure when a subscriber requests a connection to the network. The connection is considered virtual because no direct physical path exists between the source and destination of a connection, but only appears to from the connected party's perspective. The two fields in a cell's header, the Virtual Path Identifier (VPI) and the Virtual Channel Identifier (VCI) combine to form the cell identifier and indicate which virtual connection to transport the cell over. Both the VPI and VCI have only local significance and are translated at each network node. Thus, to setup a virtual connection, each switch in the connection is provided routing table information which identifies an incoming cell by its VPI/VCI, routes it to the correct switch output port, and translates the cell's VPI/VCI to correspond to a routing table entry in the next switch of the virtual connection. As implied by the use of two parameters, a virtual connection is formed by two types of connections: virtual paths and virtual channels. Figure 6 illustrates the relationship between virtual paths, virtual channels, and the actual transmission media.

![Diagram of Virtual Paths, Virtual Channels, and Physical Paths](image)

**Figure 6** Relationship between Virtual Paths, Virtual Channels, and Physical Paths [5, p. 284]

The VCI header field identifies a virtual channel (VC) to which a cell belongs [2, p. 31]. A VC is a simple unidirectional connection from one ATM switching node to another. It is identified by 16 bits which means any switch in the ATM network can support 64k simulates VC connections given enough internal resources are available. Furthermore, VC connections are dynamically allocatable. As a cell travels through an ATM switch, the switch translates the incoming cell's VCI to the VCI
associated with the output port of the switch and entry into the next switch's VCI routing table. When a VCI translation occurs, the connecting points where it occurred is referred to as a VC link. A continuous series of VC links is called a VC connection (VCC). One main advantage of virtual channels is that it supports multi-component services like videophony [1, p. 85]. Each component of the service can be added as needed by requesting another VC. For instance, the videophony connection can begin by establishing a voice connection over a single VC. Later, the video component can be added to the connection through another VC. Cells in any of these connections retain their sequencing.

The VPI field identifies a virtual path (VP) to which a cell belongs [2, p. 32]. A VP identifier is defined by 8 or 12 bits, depending whether a cell is at a NNI or UNI, thus 256 to 4k simulates VPs can be defined at a switching node. Unlike the VC, VP connections are statically allocatable because they are established on a semi-permanent basis by network management or user signalling. Essentially, the virtual path, also referred to as a virtual network, groups multiple VCCs together making it possible to efficiently support a large number of connections between two points in the network by simplifying their switching and management [1, p. 86]. This also simplifies call setup procedures. The amount of processing and setup time is reduced because the group of VCCs designated by the VPI are already established connections and thus have no setup time associated with them. Only the VCs up to the connecting points of the VP need to be established. Furthermore, the virtual path concept allows separation of the different QOS classes. Time critical services like voice data could be routed over a different VP than bursty computer data to prevent traffic congestion which would negatively impact the quality of the voice data. As with the VCI, the VPI is translated on a link by link basis, and VPI translation occurs before VCI translation. It is possible that neither VPI nor VCI translation is necessary, or only one requires translation. When a VP is established, it is defined to have a maximum date rate or bandwidth it is allowed to carry, emulating a physical transmission line. VCs can then be statistically multiplexed onto the VP as described in section 1.2.2.5 as long as the combination of the VCs' SCR are not greater than the bandwidth limits of the VP [6, p. 211]. Statistical multiplexing of VPs onto the physical transmission line is not permitted however. An example of routing through an ATM network can be seen in figure 7.

Figure 7 demonstrates several concepts about virtual connections and the VPs and VCs that compose them. First, several VPs have the same identification number, 2. This proposes no problem because VPI and VCI identifiers have only local significance within the ATM network. Another significant point clearly illustrated by the figure is that translation of the VPI/VCI need not occur. When subscriber Y uses VC 3 over VP 5, the virtual channel number is not translated by
node B when the connection is switched to VP 2 to reach node A. Although not shown in figure 7, the same scenario applies to VPs. Lastly, the figure shows that two virtual channels can connect to the same node without conflict because they can be distinguished by dissimilar VP identifiers, as is the case where two VCs with a VCI of 3 connect to node A over two different VPs.

1.2.3 Advantages and Disadvantages

Clearly, ATM has several advantages and disadvantages when compared to other transfer modes. One major advantage worth noting is the efficient use of network resource when compared to STM. For example, a voice connection in an STM system would be defined by a particular time slot in the transmitted data frame. During silent periods in the connection, no meaningful information will be communicated, and the time slots will be wasted. These wasted time slots could be used to transmit other, more meaningful data. In STM, these wasted time-slots can not be allocated to another connection because the position of the time-slot in the transmission frame implicitly defines the connection. In ATM, though, these slots could be used by any connection, particularly data connections which do not require periodic transfer cycles. Thus ATM can achieve higher rates of
utilization of available transmission facilities than STM. ATM has other advantages besides efficient utilization of network resources. The following list quickly recaps these advantages [2, p. 11].

- Flexibility to support existing and undefined future services
- Dynamic bandwidth allocation
- Integrated transport of all data
- Efficient network utilization
- Connection allocation not necessarily based on PCR

As with all technologies, though, ATM possesses several disadvantages which must be mentioned. Because specific resources within the ATM network are not reserved for use by a specific connection, the possibility of network congestion emerges, resulting in cell delay variations across the network and the remote possibility that cells may even be lost. To cope with this problem, the concept of QOS was developed. Implementing this concept introduces it own difficulties and complexities though.

1.3 B-ISDN and ATM

The combination of B-ISDN and ATM provides service integration at all levels, specifically at the physical transmission, user access, and traffic/switching level. To better understand how ATM and B-ISDN work together to arrive at the vision of a single service-independent network demands an examination of the hierarchical model for B-ISDN.

1.3.1 Protocol Model

![Figure 8 B-ISDN Protocol Reference Model and Layer Functions](4, p. 59)
Many communication systems are modeled using the Open System Interconnection (OSI) model composed of seven layers of abstraction used to describe different functional components in these systems. As shown in figure 8, the B-ISDN protocol model varies from the OSI model; it is composed of three planes of operation distributed over four layers of functionality [4, p. 58]. The user plane is used to transport information. ATM functionality and protocols occupy the majority of this plane. The control plane monitors and coordinates connection control functions. The last plane in the model, the management plane, is responsible for all network supervision, consisting of functions that monitor and control the ATM network to ensure correct and efficient operation. These planes exist at some level of maturity in all other networks, but the plane of most interest here is the user plane since it embodies the majority of the functional specifications for the ATM protocol required to achieve the multiple levels of integration found in B-ISDN.

1.3.2 User Plane

The user plane describes the method and protocols used to transport information over B-ISDN. To accomplish this transport, the user plane utilizes four functional layers: the higher layer protocols, the ATM Adaptation Layers (AAL), the ATM layer, and the physical layer. Each layer has a distinctive set of functions it performs to service the layer above it. This allows any layer to change to exploit new technological advancements while isolating the remaining layers from any repercussions caused by the change. For example, if technology was developed to harness gravitational waves for the physical medium of data transmission, the current physical media used in B-ISDN could be changed with little to no impact on the layer it services, the ATM layer.

1.3.3 Service Classes

In B-ISDN, user services like videophony, telephone, computer data, and fax services will be directly supported by the higher layer protocols. Since all these services generate traffic with different intrinsic characteristics, it is vital to describe these characteristics by the use of source traffic descriptor parameters, which describe either a quantitative aspect of the traffic like peak data rate or a qualitative aspect like the traffic’s source type. Knowing these parameters, the network can grant connections after verifying the availability of adequate resources to support the traffic. To facilitate the study and development of appropriate traffic description parameters and connection admission control (CAC) procedures, user services are categorized by ITU-T into four different groups based on three characteristics to describe the general type of traffic each service generates.
These groups are called service classes [2, p. 19]. The first is Class A. Class A services are connection-oriented, Constant Bit Rate (CBR) services like voice and telemetry. These services (often called real time applications) have a time relationship between the source and destination which necessitates the need for minimal delay connections. The next class, Class B, groups services with connection-oriented, Variable Bit Rate (VBR) traffic which also has a timing relation between the sender and receiver. Class B services include applications like encoded video and audio. Class C services are identical to Class B service except there exists no timing relationship, like computer data transfers and network signalling for example. The last class, Class D, describes connectionless services like data from Local Area Networks (LAN). The following table 1 summarizes the four service classes.

<table>
<thead>
<tr>
<th>Class</th>
<th>Characteristics</th>
<th>Class A</th>
<th>Class B</th>
<th>Class C</th>
<th>Class D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>CBR</td>
<td>VBR</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Connection-Oriented</td>
<td>VBR</td>
<td>Connectionless</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Timing Factor Exists</td>
<td>No Timing Factor Exists</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1.3.4 ATM Adaptation Layers

The ATM Adaption Layer (AAL) in the B-ISDN model is responsible for mapping data between the higher layer service classes and the ATM layer [4, p. 63]. The format of the data used in the higher layer services is referred to as Protocol Data Units (PDU). The PDU format varies depending on the service class from which it came. When data is travelling from a specific service class to the ATM layer, an AAL specific to that service class converts the incoming PDUs to the format used by the ATM layer, the cell. When the direction is reversed, the AAL converts the cells arriving from the ATM layer into the PDU of the destined higher layer service. Since each class of service has different specifications for its PDU, one or more AALs exist to perform the service specific data mapping function. Because the ATM layer only transports the cell format internally from network node to network node, the conversion of data between PDU and cell only needs to occur at the edges of the network. Thus, AAL functionality only happens at the point where the user accesses the ATM network, referred to as the User Network Interface (UNI). Between networking
components like switches, called Network-Network Interfaces (NNI), no AAL functions are performed.

To complete this mapping function, two sublayers are used, the Convergence Sublayer (CS) and the Segmentation and Reassembly (SAR) sublayer [1, p. 129]. The SAR layer segments variable length PDUs passed to it from the CS layer into fixed length data units of up to 48 bytes to be placed in the payload section of cells for transportation over the ATM layer. As the data in the PDUs is segmented, however, the SAR uses a portion of each cell's 48 byte payload. Since the PDU can be of varying length, a cell sequence number, a cell sequence number protection code (a miniature CRC code), and a length indicator are added. Other bit fields may be added to the cell's payload like a CRC code protecting the actual data portion of the cell's payload. Conversely, the SAR reassembles cells from the ATM layer back into the appropriate PDU using the information stored in these bit fields before passing the information back to the CS. The SAR also may manage multiplexing of multiple CS-PDUs from several applications onto the same ATM connection. The CS layer performs various higher level functions which vary depending on the service the CS supports. For CBR services (Class A), the CS layer performs functions which include source clock recovery at the destination, management of cell delay variation and cell payload assembly delay, source data structure recovery at the receiver, monitoring of lost and misinserted cells, and the monitoring of end-to-end performance of the connection. The CS layer performs similar functions for the remaining classes. Since the CS layer is service dependent, it is often broken down into two sublayers, the Common Part Convergence Sublayer (CPCS) and the lower Service Specific Convergence Sublayer (SSCS). Overall, the AALs enhance the support the ATM layer provides to the different service classes by performing the required service specific functions, allowing the ATM layer to focus on providing fast and efficient service-independent transport of data.

1.3.5 ATM Layer

The next layer and the primary focus of this thesis is the ATM layer. The main purpose of the ATM layer is to transport data cells in sequential order based on information stored in the cell's header field. The data contained in the cell must be transported across the network transparently. To complete this activity, the ATM layer must perform multiple functions. First, it must multiplex and demultiplex streams of cells from various connections into a single stream to pass to the physical layer for transmission. Next, it adds and deletes the header information to/from cells as they are passed from/to the AALs. In addition, the ATM layer also performs translation functions on information in the cells' headers to update routing information stored there as cells are routed
through the ATM switching network. Finally, the ATM is responsible for a variety of management functions and the control of information flow at the UNI.

1.3.6 Physical Layer

The bottom layer of the B-ISDN model is the physical layer, the actual methods and media used to transmit data between nodes in the network. It is defined by two sublayers called Physical Medium (PM) and Transmission Convergence (TC) [4, p.61]. The PM layer implements the correct bit timing and line coding essential to transmit and receive bits on a specific media. The TC layer performs several functions for supporting both ATM and PM layers. It generates Header Error Correction (HEC) codes on the transmission side and error code processing at the receiving side for the cell header, delineates cells from the bit stream, maps and recovers cells to and from transmission frames if required by the chosen PM layer, and performs cell rate decoupling by the insertion or suppression of idle (empty) cells to the cell stream, adapting the transmission rate of valid ATM cells to the capacity of the physical link. The physical layer of choice for B-ISDN is the Synchronous Optical Network (SONET) / Synchronous Digital Hierarchy (SDH). This is a family of rates and frame formats defined to interface to optical networks. For the most part, SONET and SDH are compatible. Table 2 lists the data rates available in the two specifications [6, p. 325]. For NNI connections in B-ISDN, the primary data rates used at both the UNI and NNI are 155.52 Mbps and 622.08 Mbps. The 622.08 Mbps access rate, however, is not symmetrical at the UNI; one direction can be 622.08 Mbps while the other is only 155.52 Mbps. On the 155.52 Mbps connection, only 149.76 Mbps are truly available for information while only 599.04 Mbps are available on the 622.08 Mbps connection.
1.4 ATM Switches

With the basics of ATM described in the previous sections, the actual implementation of the ATM network can now be considered. The key of the ATM network which multiplexes and switches cells in B-ISDN is the ATM switch. Simply put, the switch takes cells arriving at its input ports, and based on VPI/VCI in each cells header, the switch determines the physical output port each cell needs to be transported to and routes the cells accordingly. There are considerations besides the routing of cells between input and output ports, like signalling cell processing and management and maintenance functions. Together, the components of an ATM switch work together to perform the routing and buffering of cells from the user plane and the functions to support the requirements of the management and control planes mentioned in the B-ISDN model. The basic functional components of a generic ATM switch are illustrated in figure 9. These blocks may be arranged in other configurations besides the one shown in figure 9, and those configurations are completely up to the switch designer to decide as long as the switch operates according to the standards set forth by ITU-T. The following table provides a quick overview of the function of each block [2, p. 87].

<table>
<thead>
<tr>
<th>Input Module</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>(IM) receives cells from the physical layer and prepares them to routed through the CSF</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output Module</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>(OM) prepares switched cells for transmission on the physical layer</td>
<td></td>
</tr>
</tbody>
</table>
The function and characteristics of the IM, OM, and CSF will be described in more detail in the following sections. The function and characteristics of the CAC and SM modules will only be described briefly since they are not the focus of this thesis and encompass topics too broad for discussion here.

**1.4.1 Input Module**

The Input Module (IM) first task is to terminate the incoming physical layer signal. Quite often, this will be a SONET signal. Accomplishing this task involves functions described earlier in section 1.3.6 in discussion about the physical layer of the B-ISDN protocol reference model. These functions include converting the physical signal (be it optical or electrical) into a signal the switch can process, recovery of the digital bitstream, processing any frame overhead (present if SONET is being used), determining cell boundaries, and discarding idle cells from the bit stream. Because these functions belong to the physical layer, they will not be discussed further.

After cell delineation from the digital bit stream and error checking and correction based on the HEC, cells are ready to be processed at the ATM layer. The first block encountered is the Usage Parameter Control/Network Parameter Control (UPC/NPC) block, which monitors and enforces traffic parameters for each established VCC/VPC [2, p. 128]. Cells discarded because of violations of VCC/VPC traffic parameters are counted as are cells with header errors. These counts can be examined by the SM at a later time for evaluating the switch’s performance. This block also performs traffic shaping in the case were bursty traffic threatens to cause switch congestion. Traffic
shaping involves reducing the PCR or burstiness of VBR traffic by spacing cells out. This helps reduce the number of cells discarded by the UPC due to VPC/VCC traffic contract violations. The user may also perform traffic shaping to conform to the established contract.

![Figure 10 Input Module Functional Blocks](image)

Figure 10 Input Module Functional Blocks [2, p. 98]

Once cells pass through the UPC/NPC unit, they enter the cell processing unit next. Since signalling and management cells are merged with user cells in the cell stream, the processing unit first filters out and diverts these cells to either the CAC or SM module if required, otherwise, the cells pass on to the header translation unit. This unit extracts the VCI/VPI from each cell header and accesses the VC/VP connection database of the switch. This database is essentially a routing table. From the database, the header translation block receives information about the switch output port, type of cell, delay priority, broadcast/multicast identifiers, and other data about the given cell. Also, a new VPI/VCI for the next link in the connection is returned to replace the VPI/VCI currently in the cell's header. Finally, the input module reserves the option to add additional bits to each cell for internal purposes. For instance, a 32x32 port switch may add 5 bits ($2^5 = 32$) to the cell to identify which output port on the switch the cell is destined for and to provide a tag for traversal through a self-routing CSF. Other additional bits may be added to store cell sequencing numbers or time stamps. Regardless of what information the bits store, they are for internal use only and are stripped from the cell at the output module before being transmitted on the next physical link. Figure 10 depicts the operation of the input module.

1.4.2 Output Model

The Output Module (OM) may be considered the dual of the input module, executing many input module functions only in reverse, see figure 11. In general, the OM prepares outgoing cells for physical transmission. Like the input module, several of these functions like frame generation, HEC
field generation, and cell rate decoupling belong to the physical layer of the B-ISDN model and will not be discussed further.

Overall, the OM is much simpler than the IM because it lacks the complex UPC/UNC functional block [2, p. 135]. To prepare cells for transmission, the OM reverses several ATM layer operations performed by the IM. First, signaling and management cells removed from the cell stream are reintegrated into the stream. The OM next removes all internal tags attached to a cell. If any processing based on the tag information needs to occur, it is done here. For instance, a time stamp bit field might be used to calculate the time needed to route the cell through the CSF. This information can be stored and used by the SM for performance analysis and modification to switch resource allocation guidelines.

![Output Module Functional Blocks](image)

Figure 11 Output Module Functional Blocks [2, p. 134]

As part of the housekeeping functions, the OM must perform VPI/VCI translation on certain cells. This operation is only for broadcast/multicast cells. When cells requiring multicast/broadcast enter the switch, the IM marks them specifically as broadcast/multicast cells to signal to the CSF that these cells require duplication. Performing the VPI/VCI translation at the IM would result in the duplicate cells created by the CSF all possessing the same VPI/VCI. Therefore, this function occurs only at the OM. Based on the output port number, when a multicast/broadcast cell arrives at the OM, the appropriate VPI/VCI can be queried from the VP/VC database and inserted in the cell header. After header translation functions, exiting cells are counted to support the unpleasant activity of account management for customer billing services. Finally, cells are passed on to the TC layer of the physical layer for transmission.

1.4.3 Call Admission Control Module
The Call Admission Control (CAC) module performs and controls functions related to the establishment, modification, and termination of VC and VP connections to the switch. The following list summarizes some of main functions and activities the CAC module executes [2, p. 90].

- Generation of UPC/UNC parameters
- Admission/rejection decisions on requested VPC/VCCs
- Allocation of switch resources for granted VPC/VCCs
- Negotiation of traffic parameter contracts with users requesting VPC/VCCs
- Renegotiation of traffic parameters of existing VPC/VCCs
- Interfacing to a signalling network (may be done over VCCs in ATM)

### 1.4.4 System Management Module

The System Management (SM) module performs many complicated functions to ensure the efficient and correct operation of the switch. The following list summarizes some of the main functions and activities the SM module executes [2, p. 90].

- Support of network management
- Traffic measurements and management
- Usage measurements of switch resources for account billing
- Physical layer operations and maintenance functions
- ATM layer operations and maintenance functions
- Security and management of switching and information databases
- Configuration of switching resources

One item in particular, Operations and Maintenance (OAM) functions, refers to the functions performed by the management plane which spans all layers in the B-ISDN model described in section 1.3.1. ATM layer OAM functions are responsible for monitoring the status of the ATM network and detecting and responding to network errors and malfunctions through maintenance functions.

### 1.4.5 Cell Switching Fabric Module

The Cell Switching Fabric (CSF) is often referred to as the central component of the ATM switching system. Its principal objective is to buffer and route cells from \( N \) different input modules to \( N \) different output modules. For terminology purposes, the ATM switch is commonly referred to as a \( NxN \) switch. Also, the input and output modules are referred to as input and output ports. The routing through the switch is based on an internal routing tag attached to each cell by the input module. Typically, the CSF automatically routes cells according to the tag and no internal path
calculations are needed. Because of the importance of the CSF, both the term ATM switch and CSF will be used interchangeably for the remainder of the thesis. Accordingly, the phrase switch architecture will be used to refer to the design and characteristics of a specific CSF and not the architecture of the entire switch of which the CSF is simply a subcomponent as depicted back in figure 9. Thus, the term ATM switching system will be used to encompass all the functional blocks required to implement a fully functional ATM switch. Because of its importance to the operation of B-ISDN and the ATM network, the ATM switch (CSF) and its characteristics is the focus of this thesis.

1.4.5.1 Design Factors

Many factors and requirements impact the design of the CSF. One such factor is the rate of information transfer. As already introduced, typical connections to the ATM network will occur at two defined rates: 155.52 Mbps and 622.02 Mbps. These rates are necessary to support the future broadband services like HDTV transmission. Whatever the information to be carried, the switch must be designed to handle the maximum incoming data rate. This, however, does not require the switch to internally operate at the maximum data rate. Future broadband services will not only require high data rates, but will demand broadcast and multicast functions from the switching network. For instance, the distribution of consumer video and e-mail will require a single signal or message to be distributed to multiple end-users. This implies the cells transporting this data must be copied and routed accordingly. In addition to the other two concerns, the switch design is constrained by a variety of network performance parameters used to establish different classes of QoS. The following list describes these parameters [2, p. 145].

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput</td>
<td>output traffic rate, commonly normalized with the input traffic rate</td>
</tr>
<tr>
<td>Utilization</td>
<td>average traffic input rate divided by the maximum total output rate</td>
</tr>
<tr>
<td>Switch Delay</td>
<td>the time for a cell to pass through the CSF</td>
</tr>
<tr>
<td>Cell Loss Rate</td>
<td>the rate at which cells are lost due to buffer overflows in the switch</td>
</tr>
<tr>
<td>Buffering</td>
<td>the amount of cell buffering needed to attain performance goals</td>
</tr>
<tr>
<td>Complexity</td>
<td>issues related to physical implementation, like the number of transistors</td>
</tr>
</tbody>
</table>

Two of these parameters, cell loss and switch delay merit further explanation. In ATM switch design, cells are asynchronous time division multiplexed, meaning that cells destined for a particular output port do not fill a specific time slot in a frame structured cell stream. Thus, the possibility exists that multiple cells destined for the same output port may arrive simultaneously at the switch. This situation can occur over the span of several consecutive cell arrival periods. In this case two scenarios arise. First, the switch may contain the internal resources to buffer the arriving cells
pending output processing and transmission. If the number of cells exceeds the available buffer space, though, cells will be discarded. Simply increasing the buffer space or lengthening the appropriate queue can reduce this probability, but it will not totally eliminate it. In some switch designs, internal resources may not exist to route or buffer more than a predetermined number of simultaneously arriving cells to a common output port. Similarly, cells will be discarded. When cells must be discarded, the cell discarding procedure uses the CLP header bit, which indicates the cell loss priority of the cell, to make the appropriate selection. The rate at which cells are lost is naturally referred to as the Cell Loss Probability (CLP). The design of the ATM switch must provide sufficient buffer and internal resources to minimize CLP. A typical CLP of an ATM switch ranges between $10^{-8}$ and $10^{-11}$; future broadband services require CLPs ranging between $10^{-3}$ and $10^{-6}$.

The time to switch cells through the ATM switch has significant influence on the quality of service that a switch can provide. Certain real time services, like telephony and videophony, specify maximum routing delays in order to deliver clear voice communications. Several delay components exist in the ATM network: Packetization Delay (PD), Transmission Delay (TD), Depacketization Delay (DD), and two other delays [1, p. 64]. The other two delays are caused by the ATM switch. The first delay is referred to as Fixed Delay (FD). The fixed delay describes the time a cell takes to travel from the input port to the output port when all switch resources are available, the zero load condition. This parameter is implementation-dependent. The other delay component is called Queuing Delay (QD), caused by the time cells spend in queues (buffers) awaiting output from the switch. Queueing delay has an exponential relationship with the amount of traffic imposed on the queue, and usually exceeds the fixed delay under moderate traffic loading. Because the number of cells in the queues/buffers varies depending on the type and amount of traffic the switch is processing, the queuing delay also varies. The culmination of this variation in delay across the network is referred to as cell delay variation or jitter. In ATM switch design both cell delay and delay variation must be minimized. To achieve minimal cell delay and delay variation implies the use of small buffers, which conflicts with using large buffers to create extremely a low CLP. Typical switch delay values fall in the range of several hundred microseconds.

Another major concern in switch design is fault-tolerance. Because of the amount of traffic a single switch can potentially process, a fault-intolerant design can disrupt multiple VPC/VCCs in the event of a single, but significant switch fault. Therefore, ATM switches must be designed to adapt to internal faults to minimize the disruption of communication connections. Two additional switch design concerns are the amount of buffering required and the complexity of the design. Ideally, a switch should be designed with sufficient buffering to support any traffic scenario and to eliminate...
the probability of any cell loss due to buffer overflow. Physical and practical limitations negate this possibility, and great care must be exercised to determine the amount of buffering required to achieve desired performance levels. Finally, the complexity of a switch design directly affects its physical implementation. In any design throughput and utilization should be maximized while simultaneously minimizing factors such as cell loss rate and cell delay. At times, these two design goals conflict.

1.4.5.2 Architectures

After examining the considerations which influence the design of the cell switching fabric, different cell switching architectures can be explored. Four classical switch architectures designs exist. Virtually all modern ATM switch designs contain elements of these four basic architectures, therefore, they warrant some attention. The four architectures are shared memory, shared medium, fully interconnected, and space division. The following sections will introduce the basic concepts behind each architecture.

1.4.5.2.1 Shared Memory

The first switch design is the shared memory design [4, p. 165]. The architecture consists of four main functional blocks: input module, output module, memory, and a memory controller as depicted in figure 12. Cells, after being converted to parallel bytes from the bitstream are written into a central memory unit. The internal routing tags added by the input module are redirected to a memory controller. The controller uses the information to determine the sequence of cells to read out of memory. The cell stream output from memory is then demultiplexed to the output modules for final processing and transmission.

Essentially, this technique is a central queuing process almost identical in nature to an output queuing process. In the output queuing process, cell buffers are located prior to the output module. In a shared memory design, all the buffers, once dedicated to a specific output port in the output queuing scheme, are pooled together into a central memory. The entire buffer memory is then accessible to each output port. Like output queuing technique, the shared memory design is capable of achieving a normalized throughput of one under full load traffic conditions. The shared
memory technique, however, has one additional benefit. Since all the buffers physically belong to a common buffer pool, any output port can utilize any amount of buffers for cell storage as long as there are buffers remaining in the pool. Therefore, an output port experiencing a high traffic rate can temporarily draw upon extra buffers in the shared pool at the expense of reducing the buffers available to the other output ports, which are not under heavier traffic loads. Because of this buffer sharing, the total buffer memory required to achieve a given cell loss probability can be significantly diminished over an equivalently designed output queuing method. In addition, the shared buffer memory possesses more flexibility for accommodating a wide variety of traffics, especially large bursts of traffic to a given output port.

There are several disadvantages to this technique though. First, the central memory controller is very complex. This complexity is increased if functions such as broadcast and multicast must be implemented. Second, unless specialized memory allowing multiple, simultaneous cell read/write cycles is used, the memory must operate at a speed $N$ times greater than the input data rate (150 Mbps/622 Mbps. This will be termed $R_I$, where $N$ is the number of ports on the switch. This will allow each cell input during a given time period to be written to memory before the next batch of cells arrives in the next time period. Thus, the larger the switch grows, the faster memory access must become. Memory access time has physical limitations, however, which limits the switch's maximum throughput to below $N \times R$ [2, p. 147].

1.4.5.2.2 Shared Medium

Another classic switch design in the shared medium architecture. In this architecture, all cells are routed through the switch over a common medium like an internal, TDM bus [4, p. 166] as portrayed in figure 13. When a batch of $N$ cells arrives at the switching fabric from the input modules, they are, in turn, broadcast over the shared medium. Each output port uses address filters to allow cells marked with a destination tag specifying that output port to pass through into the output port. The output port then performs functions described in section 1.4.2. Because of the nature of this design,
only output buffering is necessary, but since the buffers are not shared as in the shared memory design, more buffers will be required.

This design suffers from the same problem as the shared memory design, however, which is a speedup factor is required [2, p. 147]. For the switch to process all the incoming cells before the next batch of up to \( N \) cells arrives, the shared medium must operate \( N \) times faster than the input data rate, \( R \). This further implies that the address filters and output buffers must also operate at the increased speed. If the shared medium cannot achieve the throughput \( N \times R \), then input buffers will be required for the switch design to have acceptable cell loss rates. One advantage this design has over the shared memory architecture is that it naturally supports broadcast and multicast functions since each cell is already being broadcast on the internal medium.

\[ \text{Figure 13 Shared Medium Architecture [2, p. 148]} \]

1.4.5.2.3 Fully Interconnected

Another design approach is the fully interconnected switch design [2, p. 148]. In this design, each input port has a physical path to each output port. An input port broadcasts an incoming cell onto its own broadcast bus to which every output port has an address filter attached to. If the internal address tag added to a cell corresponds to the address assigned to an output port, then the address filter attached to that output port will allow the cell to pass through to the output port's buffers. The design of the architecture can be seen in figures 14 and 15.

The fully interconnected design has several obvious advantages. Like the shared medium architecture, the fully interconnected design is naturally suited to implement broadcast/multicast functions since each input port already broadcasts its cell stream to each output port. Unlike the shared medium and share memory designs, this design requires no speedup factor for the address filters and buffers since each processes data from only one input port. In addition, the design is very simple, modular, and all hardware operates at the same internal speed, making implementation much easier. Unfortunately, the design has one major drawback. Because each output port has \( X \) buffers servicing each of the \( N \) broadcast buses from the \( N \) input ports, and there are \( N \) output ports, the number of buffers grows at a rate of \( N^2 \) as the switch size is increased. Even though the
size of the switch is limited by this factor, the speed of it may be increased to the limitation imposed by the physical technology.

1.4.5.2.4 Space Division

The last switch design is based on a space division concept, exemplified by the classic crossbar switch. In this approach, a fabric of switching elements are combined to connect any input port to any output port. These switching elements may be distributed over several stages, leading to the name Multistage Interconnection Network (MIN). A classic example of a MIN is the Banyan network, shown in figure 16. In this MIN, each stage is composed of 2x2 switching elements which routes data according to some control bit. For an ATM network, this control bit would be the internal tag added to each cell to identify the output port the cell is destined for. At each stage, a specific bit in the address tag is examined. Depending on the value of that bit, the data stream is routed through either the upper or lower branch of the switching element. The stages of a MIN can easily be replicated to construct larger and larger networks.

Often, as is the case for the Banyan class of MINs, one significant problem exists. Because the network of switching elements is not fully interconnected so that each input port may communicate simultaneously with each output port, the possibility exists of a blocking condition...
occurring [8, p. 218]. Two cells destined for different output ports can collide in a switching element

Figure 16 Space Division Architecture
several stages into the MIN. This is referred to as internal blocking. Several solutions exist to remedy this problem. Sorting networks, like a batcher bitonic sorter, may be added prior to the MIN to arrange cells based on the address of the output port the cells are destined for. By sorting the cells input to the Banyan network, internal blocking can be avoided. If, however, more than one cell is destined for the same output port, other measures must be taken. Input buffers may be added to the front of the MIN to hold back one or more cells destined for the same output port. In addition, the buffers could actually be distributed within the actual MIN fabric at each switching element. If two cells arrived at a switching element together and both requested to traverse the same branch of the switching element, one cell could be temporarily buffered while the other was routed through the desired branch. Many of the issues related to space division switching, and the Banyan class in particular, will be discussed further in section 2.2.

1.4.5.2.5 Switching Element
The basic building block of the space division and of other switching architectures is the switching element. It is referred to by many names, but is most commonly referred to by the names contention switch, crossbar switch, and switching element [6, p. 102]. It executes one simple function, route two inputs to two outputs based on predefined (or possibly dynamic) decision logic. This logic in the case of ATM would involve examining the activity bit of a cell and checking an internal routing tag added to the cell header. It is possible that both inputs to a switching element may contend for the same output line, in which case the switching element must provide contention resolution through a store-and-forward buffering mechanism.

Figure 17 2x2 Switching Element [6, p. 103]
The switching element, as can be seen figure 17, is composed of decision circuitry which operates on the cell header, a latch to store the result of the current decision for the length of the cell, a shift register to delay cells while their headers are being processed, and 2x2 cross connect device. The cross connect device can be set to one of two states, cross or bar. When the cross connect is in the cross state, the left input is routed to the right output and the right input to the left output. In the bar state, the left input is routed to the left output and the right input to the right output. A schematic of a proposed cross connect device is also shown in figure 17. When the control signal is asserted, AND gates 1 and 4 are enabled while AND gates 2 and 3 are disabled, creating the bar state. When the control signal is deasserted, the reverse true. AND gates 1 and 4 are disabled while 2 and 3 are enabled, creating the cross state.

In conclusion, ATM is a transport protocol that integrates all traffic at the transmission, switching, and user access levels. Over the past seven or eight years, numerous research studies and publications have been devoted to the development of new and faster ATM switching architectures. The ATM architectures to be explored in this thesis are presented in the next chapter.
2 ATM Switch Architectures

As was introduced in the first chapter, ATM is the multiplexing and switching technique chosen to transport data over B-ISDN. To accomplish this transport, ATM must perform many functions. The point of interest in this thesis is to examine the ATM switch itself, which must properly route and buffer incoming cells while attempting to realize the potential performance gains inherent to ATM concept. As has been seen, the ATM switch is composed of several components including the CAC, SM and CSF modules. Entire papers are often written on concepts and ideas forming only small portions of these modules. To limit the scope of this thesis, only the cell switching fabric will be considered. When investigating cell switching fabrics, another problem then emerges. Dozens and dozens of papers have entered the main stream technique literature describing new ATM switching architectures within the last eight years. Unfortunately, there exist no standards forms or procedures for relating the performance of the new architectures. Many papers are written so as to highlight the strengths of the architecture that is being discussed while disguising its weaknesses. This thesis aims to examine several architectures described in literature and to perform a comparative analysis of those architectures to demonstrate their relative strengths, weaknesses, and performance under different types of traffic conditions.

To initiate this process, many architectures published in technical journals were examined. In the end, three architectures were chosen to be compared while several were discarded from the further consideration. The three chosen architectures are the Knockout, the Sunshine, and the Helical switch. Each of these architectures in general can be categorized into one of the four classic CSF described in section 1.4.5.2. The Knockout switch was chosen because of its simple yet powerful design and its modularity. The Sunshine switch was chosen because it incorporates the most comprehensive combination of ideas formed over years of study and research in the MIN field. Finally, the Helical switch architecture was chosen to study because of its recent introduction in the ATM field and the resulting lack of research into its performance, and because it introduces a rather intriguing approach to cell switching and buffering. The description of these three architectures will be given in the following three sections. In the last two sections of this chapter, some other architectures that were also considered but were not selected for further exploration are briefly described, like the Starlite architecture and several shared memory architectures. In addition, these two sections quickly explain the reasons for discarding these architectures from consideration.
2.1 Knockout Switch

2.1.1 Overview

The Knockout Switch (KOS) is the first cell switching fabric to be examined in this thesis. It was introduced in 1987 by Yu-Shuan Yeh and colleagues at AT&T Bell laboratories [9]. The KOS is a symmetrical \( N \) input by \( N \) output packet (cell) switch where cells arrive and depart in a time-slotted fashion, as seen in figure 18. All inputs and outputs ports operate at the same bit rate. It is based upon the fully interconnected topology, prefaced in section 1.4.5.2.3, and it utilizes an output queueing strategy for cell buffering. It does, however, introduce several significant design differences described in more detail in the subsequent sections. By using a fully interconnected topology, each input has its own dedicated path to each output, preventing internal conflict between two cells destined for different output ports. The only point of congestion and possible contention of internal resources is at the output of the switch. When multiple cells arrive at an output simultaneously, they contend for limited buffering resources. One cell will be serviced at the output port while the remainder are buffered based on the FIFO queueing discipline. Any newly arriving cells in successive time slots will also be queued until the backlog of buffered cells can be processed by the output port.

![Figure 18 Knockout Switch Concept [9]](image)

The major drawback with this architecture when employing output queueing is that number of cell buffers required by the switch to guarantee specific cell loss objectives grows by a quadratic factor of \( N^2 \). As \( N \) becomes larger and larger, the problem is exasperated. Using an input queueing strategy, as illustrated in figure 15 on page 30 alleviates this problem by reducing the growth factor to a linear relationship. However, the realizable throughput dramatically decreases from the 100% throughput attained with the output queueing technique. This is due to the Head of Line (HOL) problem. If a cell at the head of an input queue is blocked from exiting the queue because of routing contention with another cell at a different input queue, all the other cells behind it are also blocked even if no routing contention exists for them. Assuming that the HOL cells at the \( N \) input
queues are equally likely to be destined for any output port $j$ and that the output addresses of the HOL cells are independent of each other, the probability that one or more of the $N$ HOL cells are destined for a particular output port $j$ can be expressed as the sum of several binomial probabilities.

$$P = \sum_{k=0}^{N} \binom{N}{k} \left( \frac{1}{N} \right)^k \left( 1 - \frac{1}{N} \right)^{N-k} = 1 - \left( 1 - \frac{1}{N} \right)^N$$ \hspace{1cm} \text{Eq. 1} \hspace{1cm} \text{[6, eq. 3.5.1, p. 96]}

As $N$ becomes very large,

$$\lim_{N \to \infty} \left( 1 - \frac{1}{N} \right)^N = 1 - \frac{1}{e} = 63\% \hspace{1cm} \text{Eq. 2} \hspace{1cm} \text{[(6), eq. 3.5.2, p. 96]}

This indicates that under full load traffic (each of the $N$ input queues has a cell at every time slot), an output port will receive a cell with only a 63% probability. Assuming that the complexity of the switch architecture will be measured by the number of buffers among other things, the design of the KOS attempts to reduce complexity like the input queuing technique while maintaining the throughput performance offered by the output queuing technique. The KOS achieves this by realizing a simple assumption: cells will be lost during transmission. This loss may be due to physical transmission errors, or it may also be due to traffic congestion caused by the lack of resource reservation when ATM connections are established. Because resources are not specifically reserved in ATM like they are in regular circuit-switched networks and are instead statistically shared, congestion will inevitably occur. By proper sizing of output buffers, the probability of cell loss due to congestion can be minimized. Realizing this simple fact, the KOS purposefully introduces the another source of cell loss which can be controlled and results in a dramatic reduction in switch complexity. This mechanism is known as concentration.

### 2.1.2 Component Functionality

Many components work together to form the Knockout Switch. The overall architecture of the KOS can be seen in figure 19. The KOS is a time-slotted system, and after cells are aligned with a time slot, they are broadcast from the input ports over $N$ independent broadcast buses, one for each input port. This is advantageous because a single input

---

**Figure 19** Knockout Switch Architecture [9]
port driving its own broadcast bus allows for easy implementation and a high data transmission rate when compared to a bus shared by multiple input ports. Each output port is connected indirectly to each of the $N$ broadcast buses through a bus interface unit, forming the fully interconnected switching fabric. The bus interface unit is composed of several subcomponents. The following sections describe the exact functionality of each component in the KOS by following the path a cell travels to reach the output of the KOS. The first stop on the path through the KOS switch is the input module.

2.1.2.1 Input Module

The input module of the KOS has several responsibilities [9], some of which were mentioned in section 1.4.1. First, the IM time synchronizes each arriving cell with an internal time slot. As this is being accomplished, the input module must also add an address bit field to each incoming cell to prepare it for routing through the KOS. This field is at least $\log_2 N$ bits long and specifies the physical switch output port number the cell is destined for. This is determined during the VPI/VCI translation operation in the IM. Second, unassigned (empty) cells received at the IM must be marked as so. A one bit activity flag is added to the cell header following the address field added previously. When this field is 0, the cell is considered inactive; when it is one, the cell is considered to contain valid user or network data which must be routed. Once all functions related to the IM have been completed, the IM broadcasts the current cell onto a bus dedicated to the IM.

2.1.2.2 Cell Filters

The bus interface unit is made up of several subcomponents: $N$ cell filters, one concentrator, a shifter unit, and $L$ shared buffers as seen in figure 20. The bus interface unit is the link between each output and the input broadcast buses. At the top, each bus interface unit has a row
of \( N \) packet filters connected to the \( N \) input port broadcast buses, one filter to each of the buses. At the beginning of each time-slot, the path through the cell filter is open, allowing the cells to pass through to the concentrator. As a cell enters the cell filter, the filter examines the address field in the cell header bit by bit to determine if the cell is destined for the output port to which the packet filter is connected to. If at any bit a difference is detected, the cell is prevented from proceeding to the concentrator. Any previous address bits which passed through are simply ignored by the rest of the system. The packet filter also checks the activity bit field. If the field is zero, the cell is again blocked from passing through. When a cell is blocked, the filter just outputs a zero for the remainder of the current time-slot. To simplify construction, the cell filters are implemented using the 2x2 contention switch described in section 1.4.5.2.5 except that one input and output are not connected. For clarity, the extra input and output were not illustrated in figure 20.

2.1.2.3 Concentrator

Cells which navigate through the cell filters next arrive at the inputs of the concentrator unit [9]. For a given time-slot, zero up to \( N \) active cells may arrive at the concentrator simultaneously. Let the number of arriving cells be referred to as \( k \). The concentrator is responsible for introducing the controlled cell loss mechanism which reduces the overall complexity of the KOS. It implements this mechanism by only propagating \( L \) of the possible \( N \) arriving cells, where \( L < N \), to the next component in the bus interface unit. When \( k \leq L \), then \( k \) cells will emerge at concentrator outputs \( 1 \) to \( k \) starting at the left-most output. If more than \( L \) cells arrive simultaneously, \( k > L \), then only \( L \) cells will be processed and emerge at the \( L \) concentrator output. Since the concentrator has no cell buffering capabilities, the remaining \( k - L \) cells are discarded. By properly choosing the value of \( L \), the probability that a cell is lost at the concentrator can be controlled and made to be no greater than the possibility of losing a cell elsewhere in the network, like the probability of losing a cell due to poor transmission conditions. As will be shown, \( L \) becomes independent of \( N \) to maintain a specific cell loss probability.

The design of an 8-input 4-output concentrator unit is shown in figure 21 [6, p. 123]. It is composed of two types of components, a 2x2 contention switch and a single input, single output one bit delay element. The two elements are arranged in varying numbers into \( L \) stages. All \( N \) concentrator inputs are paired together and enter into stage \( 1 \) of the concentrator, seen in the upper left corner of figure 21. The first stage operates like a simple round robin elimination tournament, with four competitions in the first row resulting in four winners and four losers. The four winners advance in pairs to the second level of the competition in row 2 where two winners are selected. The two
winning cells enter the final competition in the third row of the stage 1 where a single winner is selected. Each contention switch in stage 1 and the other concentrator stages is set to the bar state if a cell arriving at its left input is active (activity bit = 1) or to the cross state if the cell is inactive (activity bit = 0). Winning cells thus exit the contention switch on the left output while losing cells exit on the right output. The \( N - 1 \) cells which were "knocked out" of the competition in stage 1 of the concentrator proceed along the paths to the right towards the stage 2 competition. Here, the losing cells are given another chance to compete for one of \( L-1 \) remaining concentrator outputs. Up to four cells may arrive pairwise at the first row in stage 2 to compete in the two contention switches. As can be seen from figure 21, row 1 in stage 2 is drawn at the same horizontal level as row 2 in stage 1 because these two competitions occur at the same point in time. The competition of the four cells in stage 2 proceeds exactly as was described for stage 1. The "knocked out" cells from stage 2 compete again in stages 3 and 4. Any cells which lose in stage 4 are discarded by the concentrator and are lost forever. At several points in the concentrator, it is possible that an odd number of cells are competing against each other at a particular row in a particular stage, like row 3 in stage 2. To compensate for the odd number of cells, a single bit delay unit is added which defers the odd cell to the next row of competition in the current stage,
Figure 21 Construction of KOS Concentrator Unit [6, p. 124]

achieving the requiring pairing. The placement of the delay units also ensures that (1) all cells encounter an equal number of header processing delay times and (2) all cells are time aligned when they exit the concentrator. At the output of the concentrator, up to $L$ winners emerge, starting at output 1. If four cells were input on any of the concentrator inputs, they would appear on the outputs of stages 1, 2, 3, 4. If only three cells were input, they would appear on the outputs of stages 1, 2, and 3.

2.1.2.4 Shifter

The cells which successfully pass through the concentrator unit enter a shared buffer. Before reaching the buffer, though, the cells pass through the shifter unit which has $L$ inputs and $L$ outputs [9]. This unit is used to prevent the separate buffers from overflowing. Consider the case where the concentrator loads the $L$ independent buffers directly without the assistance of the shifter. Since the cells emerging from the concentrator exit on its leftmost outputs, the buffers connected to those leftmost outputs would tend to fill up faster and overflow while the rightmost buffers in
general would remain empty. This is because on average, cells will not be addressed to an output port at a rate greater than the output port can transmit. Thus, many of the concentrator inputs will be idle, resulting in idle concentrator outputs which start from the rightmost output. To prevent the leftmost buffers from overflowing while the rightmost buffers are under utilized, the shifter unit is introduced between the two units to ensure that the $L$ independent buffers are filled with cells emerging from the concentrator in a cyclical manner. The shifter is also needed to maintain the FIFO queueing discipline in the buffers when more than one cell is destined for the same output port.

The operation of the shifter is not difficult to describe. Suppose that during time-slot $t_i$, $L_i$ cells arrive at the leftmost shifter inputs from the concentrator. The shifter then loads each of the cells into the leftmost $L_i$ buffers, one cell per buffer. Prior to completing the transfer, the shifter monitors which buffer received the last cell, say $B_{L_i}$. In the next time slot, $t_{i+1}$, up to $L_{i+1}$ cells arrive at the leftmost inputs to the shifter. Continuing where it left off, the shift begins transferring the incoming cells from the leftmost input to the leftmost buffers starting at $B_{L_{i+1}}$. When the rightmost buffer $B_L$ is reached, the shifter swings back and begins transferring to the leftmost buffer $B_0$ again. In essence, the shifter is a $L$ state switch which shifts its input $0, 1, \ldots, L-1$ outputs to the right. If $S_i$ denotes the state of the switch and $k_i$ the number of cells exiting the concentrator and reaching the shifter in time slot $i$, the behavior of the shifter can be described by

$$S_{i+1} = (S_i + k_i) \mod L \quad \text{where} \quad S_1 = 0$$

The functional example can be seen in figure 22. In this example, $L$ equals eight, and five cells arrived in time slot 1, passing through the first five shifter outputs 1 to 5 and entering buffers 1 through 5. In the second time slot, four cells arrive. The shifter circular shifts the four input cells five outputs to the right and thus the cells exit on outputs 6, 7, 8, and 1 and enter the respective buffers. On the third time slot, any arriving cells would be shifted one output to the right, thus starting at output 2. The construction of the shifter can be
accomplished with the same contention switches used to build the concentrator and cell filters. A total of \((L/2) \log_2 L \times 2\times 2\) contention switches are needed, arranged in an omega network configuration as illustrated in figure 27c in section 2.2.2.1. A state machine implementing equation 3 must be used to control the network.

2.1.2.5 Shared Buffer

The last stop for cells through the KOS are the shared buffers. Each bus interface unit has \(L\) independent buffers, each of which has a capacity (depth) of \(D\) cells. The total shared buffer size \(B\) is then equal to \(L \times D\). The buffers are filled in a cyclical fashion by the shifter unit. Even though cells may be loaded into the \(L\) buffers in parallel, cells may only be removed from one of the \(L\) buffers in a given time slot. The shared buffers, therefore, act like a queue with \(L\) inputs and only one output. The outflow of cells from the shared buffers can be viewed as token controlled system. Initially, buffer 1 has the token in time slot 1. When it finally receives a cell, it passes the cell onto the output module during the next time slot and transfers the token to buffer 2. As long as a buffer is empty, when it receives the token, it will retain the token indefinitely until it has received a cell. When buffer \(L\) is ready to pass on the token, it gives it back to buffer 1. This equitable system results in several properties. First, the number of cells in the \(L\) buffers does not differ by more than one at any given time slot. Next, buffer overflows only occur when all buffers are full. Last, the buffer with the most cells stored holds the token. The amount of buffering at each output port depends on several factors, including the desired cell loss rate and the traffic arrival pattern. The topic of the KOS performance will be presented in the next section.
2.1.3 Performance

This section introduces some of the analytical methods used to determine the performance of the KOS. The overall performance is determined by two main factors: the number of concentrator outputs \( L \) and the total size of the shared buffer located at each output port. By altering these two values, the performance of the KOS as far as cell loss rate can be changed to meet proposed design specifications. Other factors such as cell delay are influenced more by the architecture design of the KOS than by the choice of different design parameters. The first parameter which will be examined is the parameter \( L \).

As was noted earlier, the introduction of an intentional cell loss probability into the KOS design at the concentrator achieves a dramatic reduction in the total switch complexity, transforming from an exponential relationship to only a linear one. The cell loss probability at the concentrator, however, must be designed so as not to be greater than the probability of losing a cell else here in the ATM network. Assuming that in each time slot a cell arrives at each input port independently with a probability \( \alpha \), and each cell has an equal chance for being delivered to any output port, the limiting probability of \( k \) cells arriving at the same output port, \( P_k \), is described by a Poisson process.

\[
P_k(\alpha) = \frac{p^k e^{-p}}{k!} \quad k = 0, 1, \ldots, N \quad \text{Eq. 3} \quad ([10], \text{eq. 1, p. 184})
\]

This probability can also be expressed as a binomial probability where \( p = \alpha \).

\[
P_k = \binom{N}{k} \left( \frac{p}{N} \right)^k \left( 1 - \frac{p}{N} \right)^{N-k} \quad \text{for } k = 0, 1, \ldots, N. \quad \text{Eq. 4} \quad ([9], \text{eq. 1, p. 1277})
\]

The binomial probability can be used to calculate the number of cells discarded at the concentrator. If \( L+1 \) cells arrive simultaneously at the concentrator, then 1 cell will be discarded; if \( L+2 \) cells arrive; two cells will be discarded, and so on. The expected number of cells discarded by the concentrator for the given time slot can then be computed as

\[
E(\text{discarded}) = \sum_{k=L+1}^{N} (k - L) \binom{N}{k} \left( \frac{p}{N} \right)^k \left( 1 - \frac{p}{N} \right)^{N-k} \quad \text{Eq. 5} \quad ([6], \text{eq. 2, p. 131})
\]

The concentrator cell lost probability \( P_c \) can then be stated as the expected number of discarded cells divided by the average number of cells arriving at the concentrator. Since on average \( pN \) cells arrive at the concentrator in a time slot of which \( 1/N \) are actually destined to the connected output port, the real arrival rate at the concentrator is simply \( p \). Thus, \( P_c \) becomes
\[ P_c = \frac{1}{\rho} \sum_{k=L+1}^{N} (k-L) \binom{N}{k} \left( \frac{\rho}{N} \right)^k \left( 1 - \frac{\rho}{N} \right)^{N-k} \]  

Eq. 6  

\[(9)\text{, eq. 2, p. 1277}\]

Taking the limit of this equation as \( N \) approaches infinity, the probability \( P_c \) becomes

\[
\lim_{N \to \infty} P_c = \left( 1 - \frac{L}{\rho} \right) \left( 1 - \sum_{k=0}^{L} \frac{\rho^k e^{-\rho}}{k!} \right) + \frac{\rho^L e^{-\rho}}{L!}
\]

Eq. 7  

\[(9)\text{, eq. 3, p. 1277}\]

Using equations 5, 6 and 7, the probability of losing a cell at the concentrator for a cell arrival probability of \( \rho = 0.9 \) is graphed in figure 23. This graph demonstrates the cell loss probability for a concentrator with a varying number of switch inputs, \( N = 16, 32, 64, 128, \) and \( \infty \), and a varying number of concentrator outputs, \( 1 \leq L \leq 12 \). Clearly, as \( N \) approaches infinity, the number of concentrator outputs, \( L \), becomes insensitive to \( N \). A value of \( L = 12 \) is more than adequate to guarantee a cell loss rate of less than \( 10^{-10} \) for any size \( N \) and any load \( \rho \). Even though the concentrator achieves extremely low cell loss rates for modest values of \( L \), the cell loss probability is also strongly affected by the size of the shared buffers at each output port.

![Figure 23](image)

Figure 23 Concentrator Cell Loss Probabilities at 90% Traffic Load for Various \( L \) and \( N \)
Before computing the probability of losing cells in the shared buffer, the arrival rate must be known. Assuming that the same uniform cell arrival rate is used as was used to analyze the concentrator's performance, the probability $q_k$ that $k$ cells arrive at the shared buffer in any given time slot can be expressed as

$$q_k = \begin{cases} P_k(a) & k = 0, 1, ..., L - 1 \\ 1 - \sum_{k=0}^{L-1} P_k(a) & k = L \end{cases}$$

Knowing this, the probability of losing cells in the shared buffer can be computed by modeling it with a stochastic processes. A stochastic process is simply a random process that changes over time. In this case, the number of cells in the shared buffer is that variable. More precisely, a stochastic process is a collection or set of random variables which are defined on the same sample space $T$, written as $\{x_n, t \in T\}$ where $x$ is a random variable at point $t$ in the sample space $T$ [11, p. 49]. The sample space $T$ may be interpreted in many ways, but for ATM switch analysis, it is considered to be the set of time slots or time instants. Furthermore, since the point $t$ is countable, only discrete-time stochastic processes will be used. To analyze the performance of the buffer, a special stochastic process called the discrete-time Markov chain will be used. A Markov chain is a type of stochastic process which is characterized by the Markov property, commonly referred to as the memoryless property. Let the Markov chain be defined as $x$ such that

$$x = \{x_n = x ; n = 0, 1, 2, \ldots\}, \quad x \in X$$

[11], eq 3.26, p. 54

$x_n$ is the state of $x$ at time $n$, and $X$ is the state space defined by

$$X = \{0, 1, 2, \ldots\}$$

[11], eq. 3.27, pg. 54

The Markov property can then be defined by the following equation

$$P ( x_{n+1} = j | x_0, x_1, \ldots, x_n ) = P ( x_{n+1} = j | x_n )$$

$j \in X, n = 0, 1, 2, \ldots$
which indicates that the probability of being in a state at time \( n+1 \) is independent of previous states at times \( 0, 1, \ldots, n-1 \) if the current state is \( n \). This greatly reduces the amount of analytical work which must be done to describe the Markov chain. Thus, the entire chain can be described by a series of one step transition probabilities, which indicate the probability of the Markov chain moving from some state \( i \) at time \( n \) to another state \( j \) at time \( n+1 \). Furthermore, it is assumed that these probabilities are independent of the time variable \( n \). This assumption further simplifies Markov chain analysis without loss of generality. The Markov chain can thus be characterized by a transition probability matrix defined as \( P = [ p_{ij}, i, j \in X ] \). Since this is a stochastic matrix, the elements of each row must sum to one. For the analysis of the shared buffer, the Markov chain shown in figure 24 is used to model the buffer's behavior. Each state in the chain represents the number of cells currently occupying the shared buffer at the end of a time slot. The total buffer size is \( B \) while the depth of each separate buffer within the shared buffer is denoted by \( D \) \((B = L \times D) \). In this chain, it is assumed that at least one cell will depart from the buffer during the given time slot while between \( 0 \) and \( L \) cells may arrive. The long term probability (limiting probability), \( \Pi_j \), of finding the chain in state \( j \) becomes independent of the initial state \( x_0 \), and the combination of all possible \( \Pi_j \) is referred to as the steady-state distribution, long-term distribution, or invariant distribution. Naturally, the sum of all the limiting probabilities must equal one. For this case, the limiting probability is defined as

\[
\Pi_j = x_j \Pi_0
\]  

\((10), \text{eq. } 12, \text{p. } 187\)

where \( x_j \) is defined as
\[
x_j = \begin{cases} 
1 & j = 0 \\
(1 - q_0 - q_1)/q_0 & j = 1 \\
x_{j-1} - \sum_{l=0}^{j-1} q_l x_{j-l}/q_0 & j = 2, 3, ..., L - 1 \\
x_{j-1} - \sum_{l=0}^{L} q_l x_{j-l}/q_0 & j = L, L + 1, ..., c 
\end{cases}
\]

and \(\Pi_0\) normalizing factor defined in equation 14 where \(D\) is the depth of each independent buffer in the shared buffer.

\[
\Pi_0 = \frac{1}{\sum_{j=0}^{c} x_j} \tag{Eq. 14}
\]

The average number of cells lost at the shared buffer in a given time slot is defined by the equation 15, which sums the probabilities that the arrival of \(k\) cells would cause a transition from some state \(i\), where \(B-L+2 \leq i \leq B\), to a state beyond state \(B\), the limit of the buffer.

\[
\omega = \sum_{j=B-L+2}^{B} \Pi_j \sum_{l=B-j+2}^{L} (i + j - B - 1)q_l \tag{Eq. 15}
\]

To calculate the cell loss probability, the average number of cells arriving at the shared buffer must be computed. This is represented by \(\lambda\).

\[
\lambda = L \sum_{k=0}^{L} (L - k) P_k(\alpha) \tag{Eq. 16}
\]

The cell loss probability at the shared buffer, \(\delta\), is then the average number of cells loss at the shared buffer in any time slot divided by the average number of cells arriving at the shared buffer for that time slot.

\[
\delta = \omega/\lambda \tag{Eq. 17}
\]
Figure 25 plots the cell loss probability at the shared buffer for various traffic loads and buffer sizes. As can be seen from the figure, choosing a buffer size of $B = 40$ or more will ensure a cell loss rate lower than $10^6$. If a concentrator output number $L = 8$ is chosen, then the 40 buffers would be distributed into 8 independent buffers each 5 cells deep.

As was described in section 1.4.5.1, the delay a cell experiences when routed through a switch is composed of two components, a fixed length delay due to the switching architecture itself and a queueing delay. The fixed length delay can only be determined through empirical data collected from traffic studies performed on the physical design of the proposed architecture. The queueing delay can, however, be approximated from the Markov chain model of the shared buffer. Since the queueing delay typically is larger than the fixed delay, it is of primary concern. The cell delay caused by the shared buffers can be calculated by using Little's formula, which states that the average number of cell in a switching system is obtained from the product of the average cell arrival rate and the average time a cell spends inside the switch.

$$\bar{n} = \bar{\lambda} \cdot \bar{D} \quad \text{Eq. 18}$$

([6], eq. 2.11.29, p. 79)

Since the average number of cells arriving at the shared buffer is already known, the average number of cells in the switch, which is actually the average number of cells in the shared buffer, must be computed. This computation appears in the following equation.

$$\bar{n} = \sum_{j=0}^{8} j \Pi_j \quad \text{Eq. 19}$$

From these two equations, the average cell delay, $D$, caused from queueing in the shared buffers can be computed.
Figure 25  Shared Buffer Cell Loss Probabilities for Various Traffic Loads and Buffer Sizes
2.2 Sunshine Switch

2.2.1 Overview

The Sunshine switch is an ATM switch which exhibits many different design techniques which vary from the Knockout switch explored in the previous section. It was introduced in 1991 by a team working for Bell labs including James Giacopelli, Jason Hickey, and William Marcus. The Sunshine switch is a symmetrical N-input by N-output, high speed, MIN, cell switch where cells arrive and depart in a time-slotted fashion just as in the Knockout switch [13]. It is based upon the space division class of switch architectures briefly described in section 1.4.5.2.4. This architecture integrates may features together [12]. High performance, low cell loss rates are achieved by a combination of internal and output buffering techniques. Parallel Banyan routing networks route cells to the output ports and their respective queues, allowing for multiple cells to be routed to any output port simultaneously. Cells which can not be routed due to internal resource contention caused by heavy traffic are recirculated back to the input to compete for routing resources with the cells arriving in the next time slot. A multiple level priority system ensures cells have access to each output. The Sunshine architecture also includes features like self-routing and a Batcher-Banyan network. Unlike the Knockout switch, each input of the Sunshine switch does not have its own dedicated path to each output, and internal congestion is possible within the Banyan network unless certain conditions are met. Several components in the Sunshine switch manipulate the order of the arriving cells in order to meet those conditions and avoid possible internal blocking common to MINs.

2.2.2 Component Functionality

The Sunshine switch architecture is quite different from either the KOS or the helical switch, which will be described in the next section. As seen in figure 26, the Sunshine architecture is composed of eight major components. As with the KOS, cell pass through these components in a time-slotted.

**Figure 26** Sunshine Architecture Block Diagram [13]
fashion. The following sections will describe the different blocks of the Sunshine architecture.
2.2.2.1 Parallel Banyan Routing Networks

The primary routing components in the Sunshine network are the $k$ parallel Banyan networks. As described in section 1.4.5.2.4, the Banyan network is a multiple interconnect network based on a single switching element. Each switching element is a crossbar based switch with two states as described earlier. Banyans are formed by arranging these switches into tree topologies where each input to the network is the root of one tree and the outputs the leaves. Figure 27a illustrates this point. Multiple trees can be joined together to form an $N$-input, $N$-output Banyan switch where all the links in the combined trees are shared except for the root. By joining the trees, the total number of required interconnection points can be significantly reduced compared to say a fully interconnected network. The group of trees, referred to as a forest, can be combined into different arrangements like a folded switching network in figure 27b and a perfect shuffle network (also called an omega network) in figure 27c.

All of these Banyan networks share several characteristics [8, p. 217].

1. The total number of cross points is less than $N^2$
2. A unique path exists between each input and output
3. Self-routing based on a header routing tag
4. Banyans are scalable because their modular structure which allows growth without modifying the underlying routing algorithm.
5. Internally blocking when two or more cells attempt to use the same link in the network
6. $N$ concurrent connections can be established, but not all $N!$ possible permutations
7. A single Banyan can only route cells with distinct output addresses

The last three characteristics are of primary concern. Many techniques are used to overcome these problems, like using $k$ parallel Banyan networks to solve the problem of routing cells destined for the same output port simultaneously. Because multiple cells can arrive at any given output, the Sunshine uses output buffering exactly like that used by the Knockout switch in section 2.1.2.5.
Using \( k \) parallel routing networks is only a part of the solution. The remaining sections of the Sunshine architecture, the batcher, trap, concentrator, selector, and recirculator work together to sort the cells in a given time slot so as to adhere to conditions necessary for proper routing through the \( k \) parallel Banyan networks.

2.2.2.2 Input Module

Before investigating the components of the Sunshine switch, the functions of the input module must be explored. Also called an Input Port Controller (IPC) in this architecture, the input module performs several critical functions on cells [13]. After removing cells from the input data stream, the IPC adds several control fields to the cell's header. The IPC first adds an activity bit, where a zero indicates the cell is empty and a one means data is present in the cell. Following the activity bit, a minimum of \( \log_2 N \) bits are inserted to encoded the physical address of the switch output port the cell is destined for. The combination of the activity and destination bits serve to provide the routing control field required by the parallel Banyan networks. The IPC then appends another field to the cell header, the priority field. This field is divided into two subfields. The first is the QOS field which distinguishes cells belonging to different service classes. Certain service classes, such as those with constant bit rate traffic created by telephony, are inherently higher priority than cells belonging to a connectionless data service because of the time correlation between the sender and receiver of the CBR traffic. The second priority subfield is the switch priority (SP) field. Assume that the cell \( c_1 \) (the first cell of a new virtual connection) arrives at the Sunshine switch during time slot \( t - 1 \). During time \( t \), cell \( c_1 \) moves through the different stages of the switch, and because of contention at it destination output port, it is forced to recirculate back to the batcher unit. The possibility then exists that cell \( c_1 \) could arrive at the batcher unit at the same time cell \( c_2 \) (the next cell of the virtual connection) arrives. If cell \( c_1 \) were recirculated again and instead \( c_2 \) moved into the output port, the cells in this virtual connection would now be out of order. Since ATM is connection oriented and cell misordering not permissible, the SP field is added to correct for this possibility. It prevents cell misordering by counting the number of time slots that a particular cell has remained in the recirculating system. Each time through the recirculator, a cell's switch priority field is increased to indicate a higher priority. In actuality, a higher priority field is encoded by a lower numerical value for reasons to be explained in the next section. Thus, increasing a cell's priority entails decrementing the SP field. Cells arriving at the input ports for the first time are always assigned the lowest SP value where all bits are ones. Therefore, a recirculated cell with a lower SP value (higher priority) would be routed before an newly arriving cell with a higher SP value (lower priority). After completing any additional processing required, the IPC passes cells onto the batcher block.
2.2.2.3 Batcher Network

As described earlier, Banyan based networks are internally non-blocking if and only if each arriving cells in a given time slot is destined to a distinct output. The first step to overcoming the blocking problems associated with Banyan routing networks is to sort the group of incoming cells. Consider the arriving group of cells to be an unordered list. After the cells have been processed by the IPCs, the list of cells moves into the batcher/trap portion of the Sunshine switch. The batcher block is based upon the batcher bitonic sort network, and it sorts the incoming group of cells based on the destination address field attached to each cell in the IPCs [13]. Cells leave the batcher network in ascending order, with the cell containing the lowest destination address leaving on output line 1 and the cell with the highest address leaving output line $N'$. For cells which happen to have identical destination addresses, the priority field is used as an extension to the destination field to increase the sorting resolution. Lower values in the priority field indicate a higher priority cell. Thus, cells destined for a common output port end up arranged in descending order of priority on adjacent output lines. In the case where a cell is not active, it is considered to have an address higher than highest possible output address. Examining an output line $i$ of the batcher block, cell destination addresses will increase as $i$ increases, and the priority of cells addressed to the same output port will decrease as $i$ increases. The outputs of the batcher block are sequentially filled, and lower addressed output lines remain unfilled in the cases where not all input cells are active. This is because the activity bit forms the most significant bit of the routing tag, and unassigned cells (activity bit = 0) will always appear to have a lower address than an active cell (activity bit = 1) with address $y$. 

- 58 -
The Batcher sorting network can be implemented using the same contention/crossbar switch described in section 1.4.5.2.5. The layout of these switches is shown for an 8x8 sorting network in figure 28 along with a sorting example. In order to sort \(2^k\) cells, where \(k\) is 3 for this example, \(k\) stages are needed. The stages in figure 28 are indicated by the dotted rectangles. The contention switches used in this network are classified into two categories. The first group of switches are referred to as up sorters. Given two input cells with different destination addresses, the up sorter will route the cell with the larger destination address to its upper branch. The other group of switches are referred to as down sorters because they route the cell with the higher destination address to the lower output branch. The arrangement of up and down sorters depends on the stage they are located in and the desired direction of sorting. The first stage consists of four 2x2 sorters, each a single contention switch which sorts two cells. The output from the first stage then passes onto a group of two, 4x4 sorters which consist of eight switches and sort four cells each. Finally, the last stage holds 12 switches, forming a single 8x8 sorter. Each sorter is composed of either up or down sorters. With the exception of the first stage where the sorters are composed of single switching elements, the interconnection necessary to form the sorter can vary. In this example a perfect shuffle interconnection is used to form the 4x4 and 8x8 sorters from the individual switching elements. A perfect shuffle topology is also used for the links between the sorters. Since the network must sort in an ascending order as described in the last paragraph, the first, or top, sorter in each stage must be a down sorter. The pattern of sorters is then alternated.
in each stage moving down the columns of switches. Thus, in the first stage, the first down sorter is followed by an up sorter, a down sorter, and another up sorter, respectively.

In general, for a \( N \times N' \) batcher sorting network, \( N/2 \) switching elements are needed per column (where \( N' \) must be a power of two) and there are \( \log_2 N' \) types of sorters. The first stage of \( N/2 \), 2x2 sorters consist of single switching elements occupying a single column in the network. The next stage consists \( N/4 \), 4x4 sorters which claim two columns of \( N/2 \) switching elements. The last stage of the network will contain a single sorter occupying \( \log_2 N' \) columns. From this, the total number of columns required is expressed as

\[
1 + 2 + \ldots + \log_2 N' = \frac{1}{2} \log_2 N' \cdot (1 + \log_2 N') \tag{Eq. 20}
\]

\( ([6], \text{p. 135}) \)

Since each column possesses \( N/2 \) switching elements, the total number of switching elements needed for this sorting network is equal to

\[
\frac{N'}{4} \cdot \log_2 N' \cdot (1 + \log_2 N') \tag{Eq. 21}
\]

\( ([6], \text{p. 135}) \)

This equation will have more significance in section 4.4.
2.2.2.4 Trap Network

The trap network follows the batcher network and resolves any output port contention which may exist among the sorted cells. If multiple cells addressed to the same output port were not trapped, internal blockage would occur when the cells reached the Banyan routing networks. This is partially over come by having $k$ parallel Banyan routing networks to the output ports, thus allowing up to $k$ cells destined for the same output port to be routed simultaneously. If there are more than $k$ cells traveling to the same output port, blocking will still occur. The trap averts this by selecting at most $k$ cells for routing to the parallel Banyan networks in a given time slot [13]. It accomplishes this by comparing the address of the cell on input line $i$ with that of the cell on input line $i - k$. If the two addresses are equal, then there are at least $k$ higher priority cells waiting to be routed ahead of the cell on input line $i$. Thus, the trap network would allow the $k$ higher priority cells to be routed while the cell on input line $i$ would be marked so as to be diverted to the recirculation unit later. To mark cells for recirculation, the trap unit swaps the priority and destination fields in the cell's additional control header. Since the priority field is defined to have a leading bit of 0, cells marked for recirculation will now appear to be inactive since the 0 bit will fill the position recognized as the activity bit. This condition will be detected by the next stage, and its importance will be explained in the next section. In addition, the trap network also sets the address field of unassigned cells to all ones to clearly identify them. The combination of the batcher and trap networks resolves any output port contention problems caused by internal blocking in the Banyan routing networks by sorting and selecting the appropriate number cells to be delivered to the Banyan networks. The trap unit can be implemented by using a single row of comparators followed by a rotate unit which swaps the two control fields [22].

2.2.2.5 Concentrator

A Banyan network is only internally nonblocking under a specific set of conditions. First, the cells input to the network must be sorted in either ascending or descending order, and second, the cells must occupy contiguous input lines. The trap network may, however, introduces gaps between the cells in the sorted list when it marks certain cells destined for same output port for recirculation. Since those marked cells will not be routed to the Banyan networks, the output lines they would have occupied sit idle and thus form gaps. To correct for this, a concentrator unit is added to the Sunshine architecture. This concentrator is not, however, the same as the one used in the Knockout switch. The concentrator in the Sunshine architecture exists to separate yet maintain the
relative, internal order of the two lists emerging from the trap network, the list of cells to be routed to the parallel Banyan networks and the list to be diverted to the recirculation unit. It regroups the two incoming lists and outputs each group onto a contiguous set of output lines. The concentrator network can be easily implemented with \( \log_2 N \) stages of switching elements. Thus, the list of cells emerging from the concentrator to be routed will satisfy the nonblocking conditions of the Banyan networks.

Assuming for the moment that the trap network does not swap cell header fields to mark cells for recirculation, an inequity will exist among the cells exiting the concentrator composed of only \( \log_2 N \) stages. As the cells marked for recirculation leave the concentrator unit, they will be presented to the recirculator in an order based on their destination address. If the recirculator becomes congested due to a momentary traffic burst, a cell with higher destination addresses could be discarded in favor of lower addressed cells simply because of the order in which they were presented to the recirculator in, regardless of whether or not the cell with the higher destination had a higher priority than the cell with the lower destination. Consider the example illustrated in figure 29. In this example, assume that the Sunshine switch has 2 recirculating lines and 1 Banyan routing network. If multiple cells are destined to the same output port in this configuration, only one of those cells will be routed through the single Banyan network while the others recirculate back to the batcher unit on the 2 available recirculation lines. Assume in time slot \( t \), that three cells destined for output port 5 arrive and the remainder of cells are unassigned. Each cell will be given

![Figure 29 The Problem with a Single Batcher Based Concentrator](image-url)
an initial priority of low. Only one of the three cells will be routed to output port 5, and the two will recirculate. When the two cells recirculate, their priority fields will be changed to medium. In time slot $t_2$, three cells destined for output port 2 (with priority low) arrive at the concentrator as do the two recirculated cells destined for output port 5. One cell will be routed to output port 2 and another to output port 5, leaving two cells with an address of 2 and another with address 5. Because of the sorted order, the recirculator unit will accept the first two cells presented to it, the cells destined for output port 2. Even though the cell with output port address 5 has a higher priority than the cells addressed to output port 5, it will be discarded.

To correct this problem, a second batcher network will be used to perform the concentrator function while at the same time eliminating this inequity. This is where the importance of swapping priority and routing field in the trap unit can be explained. Normally, the concentrating batcher assumes that inactive cells have the highest lowest addresses because their activity bits (the MSBs of the routing tag) are zero. It still, however, sorts inactive cells based on the addresses stored in the routing tags. By swapping the priority and routing header fields in the cells to be recirculated, the leading zero bit in the priority field ends up filling the position of the activity bit. These cells now appear to be inactive. When routed through the concentrating batcher, the recirculated cells will be directed to the lower addressed output lines of the concentrator, and in addition, those cells will then be sorted according to the values present in the priority field. The sorting will produce recirculated cells ordered from highest to lowest priority. Cells that are destined for the Banyan networks are unaltered by the trap and concentrator units and simply pass through. Since their activity bits are all ones, they are directed towards the higher addressed output lines of the concentrator. Starting at concentrator output line 0 and moving to output line $N - 1$, cells will emerge from the concentrator in the following order: cells to be recirculated with highest priority cells
first, idle cells, and finally cells to be routed. Figure 30 illustrates the sorting output order of cells after passing through the batcher, trap, and concentrator unit.

**Figure 30** Concentrator Cell Output Order

- PF = Highest Priority Field
- pf = Lowest Priority Field
- ADDR = Highest Output Addr
- addr = Lowest Output Addr
2.2.2.6 Selector

Once cells have been concentrated into two separate lists, the selector determines where to routing the incoming cells [13]. For every input, the selector possesses two outputs: one to the recirculators and one to the Banyan networks. The selector's routing decision is based on the state of a cell's activity bit. If it is one, the cell is moved to the Banyan networks. If, on the other hand, the activity bit is a zero, the cell is moved to one of $T$ recirculator paths. To make sure each of the $k$ parallel Banyan networks receives a set of cells with distinct output addresses, cells output from the selector are alternated among the $k$ Banyan networks. Since at most $k$ cells with the same destination address occupy consecutive output lines on the concentrator and the selector, each of the cells will route through a different Banyan network, ensuring no blocking occurs. In essence, the sorted list of cells is divided into $k$ parts, one part to be routed by each Banyan network.

2.2.2.7 Recirculator

The primary objective of the recirculator is to buffer cells and to provide a set of $T$ parallel paths back to the inputs of the batcher unit, reinserting the cell into the batcher unit in the next time slot for another opportunity to be routed to an output port [13]. Assume $j$ active cells are destined for the output port during a given time slot and $k$ parallel Banyan networks are available. If $j > k$, only $k$ of those active cells will be routed through the selector to the $k$ parallel Banyan networks. The cells denied access to the Banyan networks, $j - k$, are routed through the selector to the recirculator. If, however, $j - k > T$, then the $j - k - T$ lowest priority cells will be discarded by the recirculator. In completing the stated objective, three unique functions are performed. First, the recirculator decrements the SP priority field to increase the cell's overall priority, increasing its chances of being routed instead of being recirculated again. Second, the recirculator exchanges the priority field and destination address field in the cell's header back to their normal position, reversing the work done by the trap unit, to allow the batcher unit to properly process them. Finally, the recirculator is responsible for time aligning the arrival of recirculated cells at the batcher unit with the group of cells to be submitted in the next time slot from the IPCs. Using simple delay buffers based on FIFO queuing discipline can create this effect. After cells leave the delay buffer, they enter onto batcher input lines which are specifically dedicated to servicing the recirculator buffers. Thus, as the number of available recirculators are increased, the overall complexity of the Sunshine switch is also increased because each of these $T$ lines must be included into the design of the batcher, trap and concentrator, and selector units.
2.2.3 Performance

This section briefly describes the performance of the Sunshine switch architecture in terms of the analytical methods introduced during the discussion of the KOS performance. The overall performance of the Sunshine architecture is determined by three main factors: the number of recirculating lines $T$ back to the input of the batcher unit, the number of parallel Banyan routing networks, and the total size of the shared buffers located at each output port. By altering these three parameters, the performance of the Sunshine network can be changed to meet different levels of design specifications related to cell loss probability and cell delay.

The first factor to be considered is the output buffers. Since the shared output buffers operate identically as the buffers in the KOS, the analysis performed on the KOS shared buffers is perfectly valid for describing the behavior of the Sunshine shared buffers assuming that the same uniform traffic pattern is being considered. The only precaution is that some of the parameter $L$ used in the KOS to describe the number of concentrator outputs must be replaced by $k$, the number of parallel Banyan networks. With this accounted for, the cell loss probabilities should be identical. The cell delay calculation used in the KOS performance analysis will yield comparable results for the Sunshine network. In the KOS analysis, however, it was assumed that the queueing delay caused by the buffers contributed to the bulk of the delay experienced by the cell. This is not 100% true for the Sunshine switch. A cell in the Sunshine switch will also experience a one time slot delay each time it is forced to recirculate. Thus, a small delay component will be added to the overall queueing delay by the presence of the recirculator.

The other two factors, $k$ and $T$ can also have a big influence on the complexity of the Sunshine switch needed to attain specific cell loss probabilities and delays. They also directly influence the performance of the Sunshine switch. The recirculator-loss probability, $\lambda_T$, can be expressed by the equation:

$$\Delta_T = \frac{LT}{N \cdot p} \quad \text{Eq. 22}$$

([14], p. 455)

where $N$ is the number of output ports, $p$ the cell arrival rate per destination (load), and $L_T$ the expected number of lost cells for $T$ recirculator lines during a single cell time slot. Typically, the number of recirculator lines $T$ is normalized by the number of input ports, creating the factor
recirculators per input. The graph in figure 31 illustrates the effect of the number of recirculator lines for the uniform traffic condition and assuming only one Banyan network is present.

When more than one Banyan network is present, the performance is again altered. More Banyan networks allows multiple cells destined to the same output port to be delivered simultaneously to the port. The cells, therefore, have a lower probability of being recirculated and possibly lost when all recirculator lines are occupied. As the number of Banyan networks increases, the cell loss probability decreases significantly. When the number of parallel Banyan networks gets larger, however, a diminishing change in performance gain is realized. Figure 32 illustrates all these effects caused by altering the number of Banyan routing networks.

Figure 31  Recirculator Cell Loss Probabilities for Various Traffic Loads and one Banyan [13]
Figure 32 Recirculator Cell Loss Probabilities for 100% Load and $k$ Banyan Networks [13]
2.3 Helical Switch

2.3.1 Overview

Another interesting ATM switching architecture was recently introduced into the research community in 1994. It is called the Helical switch, developed by Indra Widjaja and Alberto Leon-Garcia \[15\][16]. The Helical switch varies greatly in design from both the Knockout and Sunshine switch, taking some rather unconventional approaches to routing cells while maintaining cell sequencing. The only similarity is that the Helical switch is a space division MIN like the Sunshine architecture. Unlike Sunshine though, the Helical switch is a multipath switch which implements the self-routing of cells with efficient buffer sharing. Being multipath means that two cells arriving at a same input port and heading for the same output port may traverse two entirely different paths within the switch. This condition raises the possibility of cell sequencing problems, however. To overcome this, the Helical switch introduces the concept of a virtual helix which forces cells routed along different paths to proceed in order and to fill cell buffers in an uniform manner. The cell buffers are distributed through out the switch in each stage (internal buffering). Because of the virtual helix, the Helical switch can prevent internal cell congestion which leads to cell loss by distributing the incoming load of cells along different paths at every stage within the switch. The Helical switch therefore proposes to perform well under bursty or nonuniform traffic when compared to switch architectures like the Sunshine and Knockout switches.

2.3.2 Component Functionality

The Helical switch only shares one component in common with the other two architectures, cell buffers. Unlike the other architecture which use output buffering, the cell buffers in the Helical architecture are distributed through out the switch. The remaining components in the Helical switch are new. As seen in the block diagram (an 8x8 switch) in figure 33, the Helical switch architecture is composed of only three main components, which are cell buffers, broadcast units, and concentrators. As with the KOS and Sunshine architectures, cell pass through these components in a time-slotted fashion. The following sections will describe the different blocks of the Helical architecture.

2.3.2.1 Input Module
Although not shown in figure 33, the input module performs two important functions on cells. After removing cells from the input data stream, the IM adds two control fields to the cell's header. The IM first adds a $\log_2 N$ length bit field to encoded the physical address of the switch output port the cell is destined for. Second, the IM inserts the cell identity bit after the address field, expressing whether the cell is a real or dummy cell. A one in this field means the cell contains active data and is real. Dummy cells on the other hand retain no useful information and are used within the switch fabric only to help preserve cell sequencing. They are coded by a zero. Unlike the two architectures, the input module of the Helical switch must detect unassigned cells and block their entrance into the switching fabric. In the other two architectures, unassigned cells were discarded at some point within the switching fabric, but not necessarily at the input module.

![Figure 33 Helical Architecture Block Diagram](image)

**Figure 33** Helical Architecture Block Diagram [15]

### 2.3.2.2 Broadcast Unit
The broadcast unit is responsible for two primary objectives. First, it aids in the routing of cells to their destinations much like the switching elements used in Banyan networks. Second, the broadcast unit prevents cells from overtaking cells that were previously processed as they move from stage to stage [15]. A 1x2 switching element similar to the 2x2 contention switch can be used to implement this component, but some modifications are needed to achieve the operation. The operation of the switching element is based upon the value of one of the destination bits in the cell's header. The bit which is examined in turn depends on the stage that the broadcast unit in question is located at. From the incoming cell, two nearly identical cells will be broadcast, one to each broadcast unit output. The only difference between the output cells is the state of their identity bit. One cell will be marked as real while the other will be marked as dummy. The determination of the identity bit of each output cell is based upon the selected destination bit. For a broadcast unit at stage \( k \), the \( k^{th} \) most significant destination bit in the cell's header will be used. If the bit is a zero, then the real cell will be broadcast from the upper output line of the broadcast unit and the dummy from the lower output line. If the destination bit is a one, the operation is reversed; the real cell emerges from the lower output line and the dummy cell from the upper one. Figure 34 illustrates the operation of the broadcast unit.

![Figure 34 Operation of the Broadcast Unit [15]](image)

**Figure 34** Operation of the Broadcast Unit [15]

### 2.3.2.3 Buffers Units

Cells which leave the broadcast units next enter cell buffers. These buffers are more simplistic than the shared buffers used at the output of the Sunshine and Knockout switches. Since each cell buffer receives cells from one specific broadcast unit, only one cell per time slot may arrive. Thus, the cell buffer operates under the traditional FIFO queuing discipline, with one input and one output. No distinction is made within the buffer between real and dummy cells. Both types are stored and processed the same. When a cell in the buffer reaches the Head of Line (HOL) position, it contends will other HOL cells in different buffers for the right to pass through a concentrator. If access is denied, the HOL cell is forced to remain in the HOL position until the next time slot when it may
compete again. When access is finally granted, the HOL cell moves to the Helical concentrator. This HOL transmission process will be described in more detail in the latter part of the next section.
Concentrators

Cells occupying the HOL position in the cell buffers also contend for access to the concentrators. The function of the concentrator is somewhat similar to the function of the concentrator in the Knockout switch. Assuming that the concentrator possesses $N'$ input lines from the $N'$ buffers servicing it, the concentrator attempts to transfer $N/2$ real cells from HOL positions in the corresponding buffers to its $N/2$ output lines while discarding as many dummy cells occupying HOL positions in those buffers. This operation is accomplished in a robin fashion. At the beginning of each time slot, an input pointer, $p_i$, within the concentrator initially indicates which of its input buffers should be examined for a real cell. From that buffer, the concentrator begins scanning each cell in the HOL position of each buffer. Each time the concentrator scans the HOL cell of a particular buffer, $p_i$ is incremented modulo $N'$. If a real cell is found, it is transferred through the concentrator and out to the broadcast unit in the next stage, which is indicated by the concentrator output pointer $p_2$. The output pointer $p_2$ is then incremented by one modulo $N/2$. If the HOL cell is discovered to be a dummy, it is removed from the buffer and discarded without being transferred to the output, and the HOL cell in the next buffer is examined. This operation continues for one time slot until one of the two following conditions is encountered [15]:

1. $N/2$ real cells are transferred and the $(N/2 + 1)$ HOL real cell has been reached ($p_i$ will point to this cell at the beginning of the next time slot), or
2. All $N'$ HOL cells have been processed, meaning $N/2$ or fewer cells were encountered. Pointer $p_i$ will point to same HOL cell in the next time slot that it pointed to in the beginning of the current time slot.

One major conclusion that can be drawn about the concentrator's behavior is that no cell loss occurs within the concentrator itself, unlike the concentrator in the Knockout Switch. An example of the operation of a 4x2 concentrator at some stage $k$ is shown in figure 35. Part (a) represents the state of the concentrator at time $t$, while part (b) represents time $t + 1$. At time $t$, $p_i$ points

![Figure 35 Operation of the Helical Concentrator](image)

- 73 -
to input line 1, and \( p_2 \) points to output line 0. After examining the contents of the buffer indicated by \( p_1 \), the concentrator finds a real cell, \( r_6 \). It transfers \( r_6 \) to the output line designated by \( p_2 \), line 0. After finding a real cell, both \( p_1 \) and \( p_2 \) are incremented to 2 and 1 respectively. When the concentrator next examines the HOL cell at the buffer pointed to by \( p_1 \), it finds a dummy cell, \( d_6 \). The concentrator discards this cell and increments \( p_1 \) to input line 3. Here, it again finds a real cell, \( r_7 \). The concentrator removes \( r_7 \) from \( p_1 \) to \( ft \), or input line 3 to output line 1. Again, both pointers are incremented. Since two real cells (\( N/2 \) real cells) were found, the concentrator has completed processing for time \( t \). If cell \( r_6 \) was replaced by a dummy cell, say \( d_6 \), then dummy cell would be removed be removed since the \( N/2 + 1 \) real cell had not be found. Because \( r_6 \) is the \( N/2 + 1 \) HOL real cell for time \( t \), the concentrator cannot remove it. The results of this processing can be seen in time \( t + 1 \). During this time slot \( t + 1 \), cells \( r_6 \) and \( r_{10} \) would be removed and output on lines 0 and 1, ending with \( p_1 \) pointing to input line 3 and \( p_2 \) pointing to output line 0.

With the operation of the concentrator explained, the importance of the broadcast unit can be illustrated in figure 36 [15]. Assume for the moment that the broadcast unit was replaced by a conventional switching element which would route cells either on its upper or lower link but not both (no dummy cells). In figure 36a, it is assumed that the empty slot was created in time \( t - 1 \) because a cell was instead routed on the lower link of the switching element which services the buffer with the empty slot via its upper link. If cells \( r_1 \) and \( r_2 \) are removed in time \( t \) and three new cells arrive, \( r_{10} \), \( r_{11} \), and \( r_{12} \), the resulting buffer state is shown in time \( t + 1 \). In this situation, cell \( r_{12} \) is now ahead of \( r_6 \) and the cells are out of sequence.

**Figure 36** Importance of the Broadcast Unit [15]
sequence. By injecting dummy cells as seen in figure 36b, the sequencing problem is totally eliminated. The empty slot in part a which caused the problem is now filled with a dummy cell created by the usage of broadcast units. The broadcast units force the buffers at the upper and lower links to be filled in a helical fashion, preventing real cells from being misordered. The combination of the broadcast unit and concentrator thus converts the $N'$ FIFO input buffers at a concentrator in some stage $k$ into a single virtual, FIFO buffer, to which cells are added and removed in a helical manner.

Returning to the concentrator unit, it may initially seem that a speedup factor may be required to implement the concentrator unit since $N'$ HOL cells must be scanned in each time slot. By using a Running Adder Network (RAN) and a nonblocking routing network, this speedup factor can be avoided [15]. The construction of an $8\times4$ concentrator is illustrated in figure 37. The running adder provides the active processing required to determine which of the $N'$ input cells should be granted routing access through the passive routing network. The operation of the running adder network is broken down into two phases for each time slot. In the first phase of the time slot, each nonempty buffer (up to $N'$ buffers) connected to the $N' \times N'/2$ concentrator makes a request to transmit its HOL cell to the RAN. Since only $N'/2$ of the requests can be granted in a time slot, the RAN computes which buffers can and cannot transmit. From this computation, control signals are sent back to each buffer from the running adder network informing them whether or not the buffer can transmit its HOL cell. In the second phase of operation, the buffers which were granted permission to transmit forward their cells to the RAN. The running adder network discards cells with a dummy identity bit and allows those with a real identity bit to pass through to the routing network.

![Figure 37](image)

**Figure 37** Construction of Helical Concentrator [15]

To determine which cells may or may not pass through the running adder network, each running adder in the RAN maintains a counter used to created a routing tag for cells [15]. These tags are
used by the routing network to direct the cells to the correct concentrator output lines. Effectively, the counters create the pointers $p_1$ and $p_2$ described earlier. During each time slot, one running adder is active while the remaining adders are passive. To clarify the discussion, the input and output lines of the concentrator will be labeled $i$ ($0 \leq i \leq N' - 1$) and $j$ ($0 \leq j \leq N'/2$) respectively, and running adder $i$ is associated with input line $i$. The counter value at each running adder at time $t$ is denoted $c_t(i)$. The position of the active adder during time $t$ represents the input pointer $p_1$ while the counter value of each adder represents the output pointer $p_2$. The value of the active adder is computed from the $i - 1$ counter in the previous $t - 1$ time slot. Each of the passive adders $i'$ computes its counter value based on the value passed from runner adder $i' - 1$. Assuming that the running adder $m$ was the last adder to accept a cell in the previous time slot $t - 1$, its counter value can be expressed by the term $c_t(m)$. If $r$ buffers request to transmit cells through the RAN, the method of determining the active counter value can be expressed by

$$c_t(m) = \left( c_{t-1}(m) + I_{m} \right) \mod \frac{N'}{2}$$

Eq. 23

([15], eq. 1, p. 2621)

where $m_i$ denotes the active running adder after running adder $m$, and $I_{m}$ denotes the value of the identity bit for cell arriving at running adder $m_i$. The value of the passive counters can be computed by

$$c_t(n) = \left( c_t(n-1) + I_n \right) \mod \frac{N'}{2} \quad \text{for} \quad n = m_2, m_3, ..., m_r$$

Eq. 24

([15], eq. 2, p. 2621)

If the identity bit of the requesting cell is a one (real), the routing label added to the cell is simply equal to counter value of running adder $n$ that the cell is passing through, which is termed $c_t(n)$. When the identity bit is a zero, a null tag is added to the corresponding cell and it is discarded by the receiving running adder before reaching the routing network. The routing network is a reverse Banyan with two parallel, internal links. The Banyan network uses these tags to route the incoming cells to the correct concentrator output.

2.3.2.5 Routing Example

A small routing example of cells traversing a 4x4 Helical switch is provided in figure 38. For example consider the path travelled by the cell arriving at input 0. The cell arriving there is destined for output port 3. Since the cell is at stage 1, the broadcast unit receiving the cell examines the first
most significant bit in the cell's routing field. Since the bit is a one, a dummy version of the cell is routed on the upper link of the broadcast unit while the real version moves down the lower link to the buffers servicing the lower concentrator in stage 1. In the next time slot, the cell will be passed from the concentrator to a broadcast unit in the second stage. This time the second most significant bit in the cell routing field will be checked. Again, a one forces the real copy of the cell to be forwarded down the lower link to the buffers and concentrator which service output port 3. In the final time slot, the cell will be removed by the bottom most concentrator and passed through to the output port 3.

![Routing diagram](image)

**Figure 38** Routing of Cells through a 4x4 Helical Switch

### 2.3.3 Performance

This section introduces some of the analytical methods used to determine the performance of the Helical switch. Unlike the Sunshine and Knockout switches, the only design parameter in the Helical switch is the size of the internal cell buffers. The size of the input buffers servicing a concentrator can be varied at each stage of the switch. Let \( B_k \) denote the depth of a buffer at stage \( k \) of the switch. The entire set of buffer sizes can be expressed as \( B = \{ B_1, B_2, ..., B_n \} \), where \( n = \log_2 N \) (the number of stages in the switch). By altering these sizes, the Helical switch can be made to accommodate most types of traffic conditions and meet the extremely low cell loss probabilities required by future B-ISDN services. In general, the size of the buffers increases as the last stage of the switch is approached. This is because the concentrators at the last stage tend to deny routing access to a higher percentage of HOL cells because of its small number of inputs,
creating a bottleneck. More buffers in the last few stages of the switch can be used to overcome this bottleneck, however, the total number of buffers that can be used is limited by the restrictions of physical implementation.

The study of the Helical switch under uniform traffic is similar to that of the other two architectures. Assuming that cells arrive at the inputs of the Helical switch with a probability $p$ independently of each other, a discrete-time Markov chain model can be used to represent the input buffers at each stage of the Helical switch. The two primary stochastic variables used in the analysis of the Helical switch are $X_k$ and $H_k$. The total number of cells in the buffer at stage $k$ just prior to the start of the $t^{th}$ time slot is defined as $X_k(t)$. The type of cell (R=real, D=dummy, or $\emptyset$ = unassigned) at the HOL position in the buffer at stage $k$ just prior to the start of time slot $t$ is described by $H_k(t)$. These variables are defined by the following equations.

$$X_k(t+1) = \min (X_k(t) - D_k(t), A_k(t), BN'), \quad \text{for } t = 0, 1, \ldots$$  

$$H_k(t+1) = \begin{cases}  
F_k(t) & (X_k(t) - D_k(t)) > 0 \\
G_k(t) & \text{otherwise} 
\end{cases}$$

where $BN'$ = the size of the buffer,

$D_k(t)$ = the number of cells removed by the concentrator from the buffer at stage during time slot $t$.

$A_k(t)$ = the total number of cells arriving at the buffer at stage $k$ during time slot $t$.

$F_k(t)$ = the type of cell (R, D, or $\emptyset$) at the head of the buffer at stage $k$ (excluding new arrivals) just after all removals by the concentrator at time $t$.

$G_k(t)$ = the type of cell (R, D, or $\emptyset$) of the first arrival in the buffer at stage $k$ during time slot $t$.

Thus the sequence $\{ (X_k(t), H_k(t)) , t = 0, 1, \ldots \}$ is a homogenous (time-independent) Markov chain that can be characterized by the one step transition probabilities defined for $t = 0, 1, \ldots$.

$$P^k_{(i,h),(j,h')} = P \left\{ (X_k(t+1), H_k(t+1)) = (j, h') \mid (X_k(t), H_k(t)) = (i, h) \right\}$$  

$$\left( [15], \text{eq. 13, p. 2624} \right)$$

$$(15)$$

$$(15)$$

$$(15)$$

where $k$ is the current buffer stage, $i$ and $j$ are the number of cells in the buffer, and $h$ and $h'$ are type of HOL cell (real, dummy, or unassigned) for the buffer containing $i$ and $j$ cells. The limiting
probabilities of the Markov chain that some buffer at stage \( k \) has \( i \) cells with the HOL cell being of type \( h \) can be expressed by the equation

\[
\pi_{i, h, k} = \sum_{j, h'} \pi_{j, h', k} \cdot P_{(j, h'), (i, h)}^{k} \quad 0 \leq i \leq BN', h = R, D, \emptyset \quad \text{Eq. 28}
\]

\[
\pi_{(BN', h, k)} = \sum_{j, h'} \pi_{(j, h'), (BN', h)} \cdot r_{(j, h'), (BN', h)}^{k} \quad h = R, D, \emptyset \quad \text{Eq. 29}
\]

where \( r \) is the transition probability to boundary of the buffer \((MN)\). Notice that the sum of the limiting probabilities \( \pi_{i, h, k} \) for a given \( k \) must equal one. Since the performance of the Helical switch is only concerned with real cells, the probability that \( i \) real cells will be in the buffer at stage \( k \) can be expressed as

\[
R_{(i, k)} = \sum_{j=1}^{BN'} \left( \frac{1}{2} \right)^{j-1} \pi_{(j, R, k)} + \sum_{j=1}^{BN'} \left( \frac{1}{2} \right)^{j-1} \pi_{(j, D, k)} \quad i = 1, 2, ..., BN' \quad \text{Eq. 30}
\]

Knowing the limiting probabilities, the average delay of real cells in a buffer at stage \( k \) can be determined by Little's theorem as described in section 2.1.3. The delay of a real cell is represented by

\[
T^{R}(k) = \frac{N^{R}(k)}{\lambda^{R}(k)} \quad \text{Eq. 31}
\]

where the average number of real cells in a buffer at stage \( k \) is defined as

\[
N^{R}(k) = \sum_{h=1}^{BN'} i \pi_{(i, k)}^{R} \quad \text{Eq. 32}
\]

The total delay that a real cell will experience through the switch is then simply the sum of the delay at each stage \( k \), where \( n \) is the total number of stages equal to \( \log_{2} N \).
\[ T^R = \sum_{k=1}^{n} T^R(k) \quad \text{Eq. 33} \]

\([15], \text{eq. 34, p. 2626}\)

The arrival of cells at the buffer at stage \( k \), \( \lambda^R(k) \), is described by equations 32-34 of [15]. The cell loss probability can be calculated in a similar way as performed in section 2.1.3, but modeled after the method used above to calculate the cell delay. The probability of losing a real cell in a buffer at stage \( k \), \( \omega^R(k) \), can be computed by a modified version of equation 15. The total number of cells lost throughout the switch can then be computed as

\[ \omega^R = \sum_{k=1}^{n} \omega^R(k) \quad \text{Eq. 34} \]

The total number of cells lost in a given time slot can then be divided by the arrival rate of cells at the input ports to determine the overall cell loss probability of the switch for a specific set of buffer sizes \( B \) (see equation 17).

2.4 Starlite Switch

2.4.1 Overview

Another switch considered for study was the Starlite architecture. This was the first architecture described in literature back in 1984 by Huang and Knauer which was based on the Batcher-Banyan concept explained in the Sunshine switch architecture [17]. Again, this is a time slotted switch design as were the other three switches. One advantage of this architecture is that is designed to have consistent latency. This architecture, as will be briefly explained, for all practical purposes is nothing more than a stripped down version of the Sunshine architecture, resulting in a lower overall performance.

2.4.2 Functionality
The Starlite architecture is composed of only four major components networks, a concentrator, batcher, trap, and expander (Banyan) as seen in figure 39. Two major differences exist between this architecture and the Sunshine [1, p. 216]. First, the recirculating buffers between the output of the trap unit and the input of the batcher unit are the only buffers present in the Starlite architecture. No output buffers, shared or not shared, exist. Second, the Starlite configuration only uses a single Banyan network (called an expander network in this architecture) as opposed to the \(k\) parallel Banyan networks in the Sunshine design. The combination of these two factors eliminates the possibility of being able to deliver multiple cells destined to the same output port simultaneously.

As before, the batcher unit is used to sort the list of cells arriving in a given time slot according to the routing tag attached to each cell identifying the output port the cell is addressed to. By sorting the cells, they can be routed through the single Banyan network without contention as long as each cell possesses an unique destination address. The trap unit is included in the design to cover the possibility when this condition is not true. When the trap detects duplicate cells destined to the same output port, it diverts all but one of those cells back to the inputs of the batcher unit to allow them another chance to be routed in the next time slot. The cells which pass through the trap are then routed by the Banyan network. Like the Sunshine, recirculated (aged) cells are assigned higher priority so as to avoid out-of-sequence problems. One additional component not found in the Sunshine architecture is the concentrator. This component is included based on the fact that not all input modules will produce active cells for routing in every time slot. The concentrator reduces the number of parallel routing lines between the input modules and the batcher network. This reduces the number lines required in the sorting network to less than are present at the input ports, reducing the overall complexity of the switch. This component is implemented with an inverse Banyan network which directs inactive cells to the unconnected outputs of the concentrator where they are dropped.

2.4.3 Conclusions

![Figure 39 Starlite Architecture Block Diagram](1, p. 216)
Because of the combination of multiple factors, the Starlite performance is much less than the Sunshine. To compensate for the reduced performance, the number of recirculators used within the switch would have to be dramatically increased. Typically, the number of recirculators is referenced against the number of input ports. Table 3 compares the performance of the Starlite and Sunshine switch in terms of cell loss probability for three different recirculators per input port values [13]. In this case, the Sunshine switch is using 2 parallel Banyan routing networks. Clearly, the Starlite performs much worse than the Sunshine architecture. To attain a cell loss probability of $10^{-6}$ in the Starlite switch would require more than 2.5 recirculators per input port. These recirculating buffers would have to be incorporated into the same physical package as the switching components, limiting the total number that could be realized. In the Sunshine architecture though, the use of parallel Banyan networks shifts the queuing requirements to the output ports which can utilize inexpensive, high density RAM chips for output queuing. Because of its similarity to the Sunshine architecture and its low performance, the Starlite architecture will not be studied in this thesis.

### 2.5 Shared Buffer Memory Switch

#### 2.5.1 Overview

Many shared buffer memory architectures have been introduced in literature and realized in actual silicon [18][19]. The shared buffer memory architecture shown in figure 40 has several features and characteristics which make it attractive for further study and development. The primary characteristic of the shared buffer memory architecture is similar to the output

<table>
<thead>
<tr>
<th>100% Load</th>
<th>Recirculators per Input Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starlite</td>
<td>$&gt; 10^{-1}$ $&gt; 10^{-1}$ $&gt; 10^{-1}$</td>
</tr>
<tr>
<td>Sunshine</td>
<td>$10^{-4}$ $10^{-6}$ $&lt; 10^{-10}$</td>
</tr>
</tbody>
</table>

*Table 3 CLP Comparison of Starlite and Sunshine Architectures*

![Figure 40 Shared Buffer Memory Architecture](image-url)
queuing technique used in the KOS and Sunshine switches except for one difference. The output queuing technique is advantageous because it can achieve a normalized throughput of one under a full load condition. The shared buffer memory technique can also achieve this but with an additional benefit. Since all the buffers physically belong to a common buffer pool, any output port can utilize any amount of buffers as long as there are buffers remaining in the pool. Therefore, an output port experiencing a high traffic load can temporarily draw upon extra buffers in the shared pool at the expense of reducing the number of buffers available to the other output ports, which are not under heavier loads. Because of this sharing, the total buffer memory required to achieve a given cell loss probability can be significantly diminished over the output queuing method. In addition, the shared buffer memory possesses more flexibility for accommodating a wide variety of traffics, especially bursty traffic.

2.5.2 Conclusions

Unfortunately, this efficient buffer utilization and performance under bursty traffic comes at a certain cost. First, such sharing of a common memory area requires much higher levels of memory control logic than in any of the other mentioned architectures. Other networking functions such as broadcast and multicast further increase the complexity of the already complex control logic. Also, for a $N \times N$ switch, the memory and processing functions must operate at $N$ times faster than the input data rate. This is primarily due to the fact the $N$ input cells cannot be easily written to and read from the shared memory area simultaneously. Each read and write cycle for a cell requires its own dedicated time period. Since the access time to memory is limited by physical considerations, this speedup factor restricts how large this architecture can be scaled. This problem can be eased by the use of shared buffers which are divided into several buffer memories allowing for parallel access. Another problem with studying these architectures is related to the simulator. Developing the complex logic control in the chosen simulator is not practical since the simulator provides limited logical constructs and performs these logic operations relatively slow. If it were though, study of these architectures under the chosen simulator would be hampered by extremely long length simulation runs. For these reasons, but mostly for the complexity of the memory control logic and its negative impact on simulation execution times, this class of architectures will not be studied.
3 Architecture Evaluation

As was shown in section 2 for the KOS, Sunshine, and Helical switches, analytical models for describing the behavior and performance of the switches under uniform traffic can be derived based on various modeling techniques like the Markov chain analysis. Unfortunately, the uniform traffic condition fails to adequately describe and model the spectrum of traffic conditions generated by the services to be supported by B-ISDN and ATM. Many services will generate traffic which may also be bursty in nature. Others may transmit constant bit rate data traffic. A switch's performance will therefore vary on the type of traffic it is processing. Since the uniform traffic model cannot be used to provide a complete description of a switch's performance, multiple analytical models must be developed, one for each type of traffic condition to be used to study the switch's performance. Developing analytical models, however, becomes increasingly difficult when the relationship between consecutive cells in a connection is no longer an independent one. This situation exists in many types of traffic, like encoded video traffic. The difficulty of developing these models is due to the increasing number of stochastic variables which must be used to adequately describe the traffic pattern and their propagation through an analytical performance analysis. Computer simulation, on the other hand, provides a cost-effective means of determining performance under environments where many interdependent variables are at work.

To evaluate the performance of these architectures, a discrete-time system simulator was used to examine each architecture's performance under various traffic conditions and various architectural configurations. A model for each switching architecture was created for the chosen simulator and then subjected to four types of traffic: random, constant bit rate, bursty, and imbalanced. Using a simulator has several problems. To model and simulate a Knockout switch with its number of concentrator outputs and buffer sizes set so as to achieve typical ATM cell loss probabilities of $10^{-8}$ and lower would require at least $10^{10}$ or more cells to be simulated through the switch to attain performance data of any statistical significance. Simulating this number of cells inflicts heavy computing requirements. If a 100MHz processor was used and assuming that 1000 lines of simulator code where executed to process each cell (assuming a line of code executes in one clock cycle), the simulation of $10^{10}$ cells would take over 27 hours just for a single set of performance data points. This single sample point would only represent data for one specific traffic condition (a traffic type can also have several parameters which can also vary) and one architectural configuration. This even neglects operating system overhead. With even a few variables in the model and traffic types, the amount of computing required can quickly become prohibitive. In this thesis, the
modelling parameters were chosen for each architecture to produce performance data that could be collected in a reasonable amount of time while examining several configurations of each architecture.

The following sections will describe most aspects of how the performance of the Knockout, Sunshine, and Helical switching architectures were evaluated. In the first section, the simulator used to evaluate these three architectures will be introduced. The next section will detail the simulation study plan used to gather the necessary data for comparing the performance of the different architectures. The last two sections will stipulate which ATM performance parameters were studied as well as those that were not studied and why.

3.1 Simulator

The simulator used to study the three architectures was the ProModel, version 3.0 pre-release, produced by ProModel Corporation of Orem Utah, (801) 223-4600. This is a discrete-time, event driven, system simulator typically used for simulating manufacturing systems and factory floor layouts. The simulator is generic enough, though, that it could be used to simulate ATM switching architectures. ProModel runs on the x86 based architectures or higher with at least 8 MBytes of RAM, and 30 MBytes of hard drive space, using Windows 3.1, Windows 95, or Windows NT. ProModel was capable of processing $10^6$ cells in times between thirty minutes and two and half hours, depending on which architectural model was being simulated.

ProModel was chosen because it required only a brief learning time because of its graphics user interface and provided convenient modelling constructs. It also provided flexible modelling logic to allow great diversity in the models developed. ProModel provides several basic constructs for modelling: locations, entities, arrivals, streams, and logic statements and expressions. The locations are used to model the places or components where processing occurs or data collection is needed in the model. Entities are the basic units which are processed. They enter the model, move between one or many locations, and finally exit the system. Arrivals describe the frequency with which entities enter the system, which locations initially receive them, and any logic that must be performed on them. Random numbers are used to generate different probability distributions which can be used at add variation to the operation of a locations and entities, and streams are used for selecting the starting point (seed) in the extremely long random number sequence. Finally, the simulator provides common logic and expression constructs like if then else, goto commands, variables, and arrays.
The ProModel simulator provides many useful simulation features. First, scenarios can be created which represent a specific set of model parameters. These scenarios can then be simulated multiple times (called replications or runs) using different random numbers for each replication. During the execution of simulations, the simulator can measure many characteristics about the model, like the number of entities that passed through a location, the average delay time entities experienced at a particular location, the average utilization of queues, and the total mean delay between an entity's arrival into and departure from the system.

3.2 Simulation Study Plan

The simulation plan for evaluating the three architectures was developed in three distinct phases. In the first phase, models for simulating the Knockout, Sunshine, and Helical architectures in ProModel were created. In the second phase, four different traffic types were selected, each with its own set of variable parameters. In the final phase before simulation, two different architectural configurations were chosen for each of the three models. The following three sections describe each of these three phases of development. After formulating the simulation plan, each model was simulated according to this plan under various scenarios to collect performance data. The final section will present the different traffic type parameters selected for simulation in tabular form along with a table detailing which traffic type parameters were simulated with which architectural configurations.

3.2.1 Implementation

The first phase of the simulation plan was the implementation of the simulation models. For each of the three architectures, a model was created in ProModel to simulate its cell processing behavior. Each model possessed eight input modules (ports) and eight output modules (ports). The 8x8 configuration was chosen over other common sizes mentioned in literature, like the 16x16 and 32x32 configurations, to improve the simulation speed of the ProModel simulator. After experimentation with ProModel, it was determined that increasing the number of input/output ports of the modelled ATM switch increased simulation time even when the total number of cells submitted to that model during the simulation was held constant. Since the number of input/output ports is not a factor in the measurement of the key ATM performance parameters (see section 3.3), but the total number of cells submitted to the switch model is, the 8x8 configuration was selected to obtain optimal simulation performance from ProModel.
The create the ATM switch models, locations were used to implement the behavior of the different architecture components, while entities were used to represent ATM cells. The locations in each model were arranged into 3 stages, with each location executing its own set of logic and routing rules. The first stage of each model was composed of locations used to implement the behavior of the input ports. These locations added the appropriate routing tags (called attributes in ProModel) to the arriving entities (cells) and any other header control fields which were needed, like the SP field in the Sunshine architecture. The locations used for input ports also performed logic needed to implement the cell arrival pattern for the different traffic patterns. After the entities (cells) arrived at the input ports, they were passed onto locations which modelled the cell processing and routing behavior of the different architectures. Finally, the entities, after being processed by the behavioral model, were forwarded to a set of locations acting as output ports. Here, the cells were checked to make sure that they had arrived at the appropriate output port. Figures 41, 42, and 43 are the location and processing layouts of the three models. The arrows leaving the different location in these figures indicate the routing between locations. It should be noted that locations were not used to model the three architectures at the level of the basic switching element, but only the behavior of the overall architecture. Therefore, the layout of most of the models did not correspond component for component when compared to its respective architecture. Reducing the overall number of locations and their associated processing was necessary to achieve the highest possible simulation performance that ProModel could deliver.

![ProModel Location and Routing Layout of Knockout Switch Model](image)

As part of the model layout, special locations were added to compensate for a minor problem related to one of the simulator's characteristics. The ATM switching architectures modelled were driven by the concept of time slots which cells filled. ProModel, however, is an event driven
simulator. An example of an event would be the arrival of entity at a location or a change of state in a flag variable. Because ProModel is event driven, location and entity processing in ProModel is not directly based on the simulation clock, which was required to simulate the time slot processing behavior of the different architectures. In addition, the smallest simulation time unit offered by ProModel, 10μs, was larger than the length of a time slot used with the slowest (155Mbps) data rate link, 2.7μs. Since all the architectures operate on discrete time slots, special steps were taken to make the models operate as though they were time slot driven. Each second of simulation time was assumed to be a single time slot. On every second of the simulation, a clock entity would arrive at a clock processing location in the model, which would set various flag variables. The changing of the flag variables triggered processing events in different locations in the model for that second of simulation time. This effectively converted the model processing to a clock driven method.
After creating each model, verification of the model's performance was completed through extensive testing and simulation. The ProModel simulator has the option of allowing the user to step through the execution of the simulation to monitor its progress. The user can also examine the characteristics of all locations, entities, and variables at any point during the simulation. With these
two features, several test cases were simulated step by step to ensure the behavior of the models was implemented correctly. In addition, many long term simulations were executed where the locations representing the output ports were designed to check arriving entities (cells) to make sure they arrived at the correct output port. When a discrepancy was detected, the simulation was halted and an error flag raised. Also, output results were studied and compared against expected results as a means of model verification.

3.2.2 Traffic Types

As was mentioned at the beginning of the chapter, many types of cell traffic patterns will exist in the future B-ISDN. The analytical models developed earlier to describe the performance of the architectures were developed assuming only one particular traffic type, uniform traffic. Thus, the performance predicted from the analytical models only applies when uniform traffic is applied to the switching architecture for which the models were developed. In this second phase of developing the simulation study plan, several cell traffic patterns were selected and created to determine the performance of the three architectures under various traffic conditions.

3.2.2.1 Random

The first traffic type which was chosen to simulation was the random cell traffic. This is also called the uniform traffic case. In this traffic type, cells arrive at each of the input ports of an architecture independently of each other with a probability $p$, referred to as the input load. When $p = 0$ at an input port, the port receives no active cells. When $p = 1$, the input port receives a cell in each time slot. The arriving cells in the traffic pattern have an uniform destination distribution, meaning each of the arriving cells is equally likely to be destined to any output port. In a broadband network, this type of traffic may only be representative of low data rate sources where packetization times are very large compared to the duration of a cell's transmission time over the physical transmission links. Successive cells from these sources will thus be separated by long intervals of time, and therefore will have larger interarrival times. This traffic type was chosen for simulation for two reasons. First, this was the simplest traffic type to implement in the ProModel simulator, and performance results could be generated quickly for comparing the different architectures and verifying the correct operation of the models. Since the analytical models developed for the different architectures assumed the use of the uniform (random) traffic type, the results from the analytical
models were used to help chose the architecture design parameters so that the models would have a certain level of performance which would be measurable by the ProModel simulator. After obtaining the simulations results for the random traffic pattern, they could be confirmed against the performance predicted by the analytical models. Second, the random traffic pattern was used as a reference point for choosing the different design parameters associated with each architecture, which will be explained in section 3.2.3.

To implement this traffic type in ProModel, cells were created at each input port location of the model during each time slot. Each input port location then computed a random number, $0 \leq r \leq 1$. This value of $r$ was then compared against a predetermined value for $p$ which was set at the beginning of the simulation to indicate the average load the switch was to experience. If the value $r$ was less than $p$, then the cell was assumed to be active and passed to the routing and switching portion of the model. If the value of $r$ was greater than $p$, then cell was marked as inactive and discarded at the appropriate point in the model. For the remainder of this chapter, let $R$ be the set of parameters used to describe the random traffic. In this case, $R = \{p\}$.

### 3.2.2.2 Constant Bit Rate

The next type of traffic selected for use in the simulation study was the constant bit rate (CBR) traffic type, also referred to as periodic traffic. This traffic type was selected to model services like the telephone service where digitized voice data is generated at a constant rate of 64 kbps. Simulating this traffic type was important because it indicated whether or not the three architectures could provide circuit-switched performance without assigning cells belonging to the same connection to specific time slots in a frame transmission structure.

This traffic type was modelled in two ways, both very similar but with a single difference. First, cells were grouped into a frame structure consisting of $F$ time slots [13]. Since ATM must support CBR services like telephone, ATM switches must be able to deliver performance comparable to current circuit-switched (STM) systems without the assigning cells to specific time slots within the frame structure. To simulate a worst case condition in the simulated CBR traffic, all cells inserted into the $F$ time slots were considered to be active (100% load). Across the $F$ cell frame, cell destination addresses were randomly generated and distributed into the frame (the distribution of these cells is referred to as the cell arrival pattern). Two conditions were imposed on this process. First, each input was to submit exactly $F$ cells during the cell frame, implying a saturated input load. Second, each output port would have exactly $F$ cells addressed to it during the same cell frame, similarly
saturating the output port. In one implementation of CBR traffic, this cell arrival pattern was computed once and repeated in subsequent frames. This accurately models CBR traffic at the UNI because each input port of the model appears to receive cells from multiple connections where a cell in a particular connection is destined to the same output port and arrives a fixed time interval $F$. Each input/output port, however, is not overloaded and should ideally be capable of processing all submitted cells since there is no variability in the cell arrival pattern. In the other implementation, the cell arrival pattern was recalculated at the beginning of each frame time. This more accurately models CBR traffic which might be experienced at ATM switch at a NNI. As CBR traffic traverses other ATM switches, variation in the position of cells within the frame structure may occur since

![Figure 44 Illustration of Constant Bit Rate Traffic Implementation](image)

those switches must attempt to merge the CBR traffic with other data traffic. Figure 44 illustrates the basic CBR traffic concept. Clearly, as the size of the frame approaches infinity, the CBR traffic type transforms into the random traffic type. Let $C$ represent the set of parameters used to describe the CBR traffic. In this case, $C = \{F\}$.

Implementing these two techniques in ProModel was straightforward. At the beginning of a frame, the clock processing unit in each of the three models would initialize $N$ variables in a counter array each to $F$. Random values for cell destination addresses were then computed. If the value in the counter array indexed by the computed address was greater than zero, then the address was accepted, the associated counter array variable decremented, and the computed cell address stored in the next sequential position in an arrival pattern array of $NF$ dimensions. If the counter value
indexed by the computed address value was zero on the other hand, it was discarded and another random cell address was computed. A count of the total number of cells not assigned to the arrival pattern array, initially set to $NxF$, was also maintained. By examining this count after each cell assignment to the arrival pattern array, the clock processing unit could determine when the frame was completed. During the next $F$ time slots, the $N$ input ports would then access the $NxF$ arrival pattern array to determine which output port a cell in time slot $j$ of the frame was destined for.

### 3.2.2.3 Bursty

The next traffic type selected for simulation was the bursty traffic type [13][20]. Many services in B-ISDN will offer bursty data to the ATM network like computer file transfer connections and LAN services. This traffic type is characterized by two parameters, mean length $l$ and mean interarrival time $t$. The mean length parameter obviously indicates the number of cells which belong to the active burst. The second parameter, interarrival time, has two different definitions. In the first definition, interarrival time is used to indicate the number of time slots which separate the end of one burst and the start of another on the same input link, as shown in figure 45a. In the second definition (see figure 45b), the interarrival time specifies the average number of time slots that separate cells belonging to the same burst. For example, a interarrival time of 10 would indicate that on average, every 10<sup>th</sup> cell arriving would belong to the same burst. This particular definition causes significant problems when implemented in ProModel. When multiple bursts are active at a certain input port, according to the second definition, the burst patterns could begin to overlap, and two cells belonging to two different bursts could conceivably arrive at an input port simultaneously. Let $BU$ represent the set of parameters used to describe bursty traffic, $BU = \{l, t\}$.

To simplify the implementation of the bursty traffic pattern in ProModel, the first definition of interarrival time was used. During each time slot, an

*Figure 45 Illustration of Bursty Traffic*
input port would receive a cell. Before passing the cell onto the next stage, the input port would examine two local variables specifying the mean burst length and interarrival time to determine whether or not a burst was currently active on that port. If so, the cell would be marked active and passed to the routing network, otherwise, it would simply be discarded at the input port. To make this determination, the input port would first check the interarrival time. When interarrival time variable was zero, this indicated a burst was currently active at the port. The port would continue to mark arriving cells as active, decremented the length variable each time a cell arrived. When the length variable reached zero, the input port computed the interarrival time to the next burst on that port and its length based on Poisson distributions. If on the other hand, the interarrival time had not been zero when a cell arrived, it would have been discarded and the interarrival time variable decremented.

3.2.2.4 Imbalance

The last traffic type considered for simulation was the imbalance traffic type. In all the other traffic types, each of the input ports for all three architectures is configured to receive that particular traffic type. For example, in bursty traffic each of the eight input ports receives bursty traffic. This is the same for random and CBR traffic. When considering the entire B-ISDN and the ATM switches which route data through that network, it is not practical to assume that the CAC module of an ATM switching node will only establish connections of the same traffic type to the input ports of that switch. A more realistic assumption is that many different types of traffic will be simultaneously supported on the same switch. In fact, this is the one of the fundamental concepts of ATM and B-ISDN (see sections 1.1.5 and 1.2.1). The purpose of the imbalanced traffic type is to simulate these conditions. For imbalanced traffic, each input port of an ATM switch model will receive one of the three traffic types discussed in the previous sections, random, CBR, or bursty traffic. As an example, three input ports of model may receive random traffic, one CBR traffic, and the remaining four bursty traffic. In addition, each of the three traffic types would have its own set of loading parameters which will have to be chosen. Let \( l \) represent the set of parameters used to describe imbalanced traffic, \( l = \{l_1, l_2, ..., l_8\} \) where \( l_x \) represents the traffic type \( R, C, \) or \( BU \) at input port \( x \).

For the models created for this study, \( x = 1, 2, ..., 8 \).

All though this traffic type may more accurately model the traffic which will be experienced in the actual B-ISDN, this traffic type was not simulated for many reasons. The first and most significant problem associated with this traffic type was determining how to select a representative set of imbalanced traffic cases which could be simulated out of the thousands possible. If there are 3
choices of traffic type per input port (of which there are 8), and each traffic type has an average of 1.33 parameters associated with it (1 for random, 1 for CBR, and 2 for bursty), and there are, for argument sake, 5 possible choices for each traffic parameter, then over 40,000 possible imbalanced traffic scenarios exist which can be simulated. The problem then becomes how to choose a small subset, maybe as few as 3 to 5 scenarios, from the total set of scenarios that will provide a general, yet reliable glimpse of the performance of each switching architecture in an imbalanced traffic environment. Another reason this traffic type was not simulated is connected to the first reason: how to determine which of the three other traffic types will arrive at which input ports of the switch models. This process can be random or arbitrary. If this process is random, then several other problems arise concerning implementation in ProModel or any simulator. Like CBR traffic, determining which of the three traffic types would arrive at which input ports can be determined either once at the beginning of each simulation or periodically during the course of the simulation. If the selection occurs periodically, then some simulation time period must be specified after which a new set of input traffic types will be selected. Furthermore, once the traffic type at each input port is selected, some method must be devised for choosing the parameters for these traffic types, be it random or deterministic.

Several other problems arise when trying to implement the imbalanced traffic type. One problem involves the implementation of the CBR traffic type at fewer than $N$ input ports. As the implementation was described in section 3.2.2.2, $F$ cells are received at each of the $N$ input ports during one frame, and exactly $F$ cells among the $F^*N$ cells arriving at the input ports are destined to each of the $N$ output ports during that frame. If there are fewer than $N$ input ports receiving CBR traffic, say $N_i$ ports, then $F$ cells cannot be destined to each of the $N$ output ports. Two obvious choices exist to overcome this obstacle. First, the number of cells destined to each of the $N$ output ports can be reduced to $F/N_i$. Another possibility is to distribute the $F N_i$ cells to only $N_i$ output ports. Determining how to choose these $N_i$ ports introduces yet another problem.

Besides the problem of implementing the CBR traffic type within imbalanced traffic, another concern is related to the ProModel simulator itself. Implementing the logic in the different architecture models needed to simulate imbalanced traffic will be much more complex than the other three traffic types. This will negatively impact each model's simulation time since that time increases according to the amount of logic present in the model. This problem is compounded especially in the cases where simulation times are already very long. In addition, verifying the correct operation of imbalanced traffic within ProModel will be very challenging compared to the other three traffic types. For all these reasons, the imbalanced traffic type was dropped from consideration for simulation.
3.2.3 Configurations

In the final phase of creating the simulation study plan, several configurations were established for each of the three ATM architecture models. Instead of evaluating the three architectures under static configurations, several model parameters were altered. For the Sunshine switch, three alterable parameters existed, the number of parallel Banyan networks $k$, the number of recirculator lines $T$, and the size of the output buffers $B$. Let $S$ denote the set of these parameters, $S = \{k, T, B\}$. For the Helical switch, only the buffer sizes at each stage of the switch could be altered, denoted by $H = \{B_1^T, B_2^T, ..., B_n^T\}$, where $n = \log_2 N$. Since the switch size for simulation evaluation was 8x8, $n = 3$. Finally, the Knockout switch had two parameters which could be varied, namely the number of concentrator outputs $L$ and the size of the shared buffers, $B$. Let $K$ indicate this set of parameters for the Knockout switch, $K = \{L, B\}$. Overall, a total of eight parameters existed which could be modified. As can be easily concluded, varying all these parameters during simulation to cover all the possibilities essential for an exhaustive and thorough evaluation of these architectures, however, generates too many configurations. To simulate all these configurations would have required tens of thousands of simulations hours. Instead, several specific configurations were chosen.

Because a total of four traffic types would be used to load the switching models and each traffic type would be simulated under varying parameters, only a few architectural configurations were picked to constrain the total number of simulation scenarios to be performed. The models were configured into two primary groups. The major difference between these groups was the number of buffers, $B$, present in the model. The buffer size was changed to determine how the performance of the architecture would change. From the simulation data collected for several different buffer sizes for each of the architectures, performance trends between the three architectures could be compared, confirmed, and potentially used to predict the performance trends of the architecture with larger buffer sizes even though those larger buffer configurations were not simulated. Thus, the general idea of the simulation in this thesis was to configure the three switch architectures with small sized buffers which could be simulated within a reasonable amount of time. Changing the buffer size alone for the Knockout and Sunshine architectures was not enough though. If the buffer size was increased alone, the probability of losing cells at the buffer would naturally decrease. The cell loss probability, however, would still be the same at the Knockout switch concentrator and the Sunshine switch recirculator. Therefore, the parameters $k$, $T$, and $L$ were changed so as to make the cell loss probability at those units be of the same order of magnitude as at the output buffers.
In order to provide some reference point for comparing the simulation results of the three architectures and to determine how to choose the parameter sets $K$, $S$, and $H$ for the two different buffer configurations, a reference traffic scenario was picked. For the first buffer configuration, the random traffic pattern with a load of $p = 0.5$ was selected. For the second buffer configuration, the random traffic pattern with a load of $p = 0.7$ was selected. Using the analytical models and results developed earlier in section 2, the parameters sets $K$, $S$, and $H$ were selected so that each architecture would be synchronized to achieve approximately the same levels of performance (the cell loss probability, described in section 3.3, was the primary performance factor considered during this process). After choosing values for the initial parameter set, those values were systematically altered through trial and error to make the synchronization of performance as close as possible. Since it was known that the architectures were configured to have the approximately the same performance for one traffic scenario, any performance differences experienced between the architectures in other traffic scenarios could then be attributed to the nature of the architectures and not to how their parameter sets were chosen.

Table 4 lists the parameter sets which were chosen for simulation. The first row of parameter sets was referred to as the buffer 1 configuration and the second row was referred to as the buffer 2 configuration. Initially, the number of buffers selected for the Knockout and Sunshine architectures was determined from the analytical results in section 2.1.3. From those results in figure 23, it was determined through experimentation with ProModel that a buffer size of 20 cells was the limit of the ProModel simulator. This number of buffers was used in the buffer 2 configurations of the Knockout and Sunshine switches. The selection of a 12 cell buffer for the buffer 1 configurations was chosen because it produced initial CLP results in the three architectures which were at least an order of magnitude different than the buffer 2 configurations at the random traffic reference point of $p = 0.7$ mentioned earlier. Choosing buffer sizes less than 10 tended to yield CLP experimental results which were not meaningful and difficult to comparable because the range of CLP values covered was very narrow for random traffic loads above $p = 0.5$. The selection of the Knockout switch parameter $L$ was chosen according to the analytical concepts shown in figure 25 in order to compliment buffer CLP performance. The choice of the Sunshine parameters $k$ and $T$ was related to the value of $L$ and will be explained in the next paragraph. Also, the number of buffers needed in the Helical switch was determined by trial and error according to the synchronization process described in the previous paragraph.

Several comments must be made at this point about the configurations selected for the Sunshine architecture. First, the Sunshine configurations $S = \{3, 2, 21\}$ and $\{4, 1, 20\}$ have a slight
discrepancy between them. For each of the two buffer cases, multiple parameter sets for the Sunshine architecture were created to study the effects of varying the parameters $k$ and $T$ on the switch's performance. To do so required that the size of the shared output buffers remain constant. Obviously, a one buffer difference exists between the two Sunshine parameter sets in the buffer 2 configuration. This was unavoidable because the $k = 3$ factor in the first parameter set could only support output buffers of sizes 18 or 21, multiples of 3. Since 21 was closer to the buffer size of 20 in the other buffer 2 parameter set for the Sunshine architecture, it was chosen over the $B = 18$ parameter. Another point which should be made is that the buffer 1 parameter sets for the Sunshine architecture denoted by $S = \{k, T, B\}$ were only simulated under the random traffic scenarios. During simulation, it was realized that simulating the five Sunshine cases for all traffic types would require excessive amounts of simulation time. Originally, it was planned that these parameter sets would be extended into the buffer 2 configuration by altering the parameter $B$ appropriately while leaving the other two parameters unaltered. Since buffer 2 configurations required even more simulation to acquire the necessary data than buffer 1 configurations, the problem of excessive simulation time would have been increased more. Lastly, the Sunshine parameters $k$ and $T$ were in general chosen so that $k + T$ was the same as the Knockout switch parameter $L$. By doing this, both the Sunshine and Knockout switch architectures were capable of processing the same number of cells destined to the same output port simultaneously, only in different ways.

### Table 4 Simulated Architecture Configurations Described with Parameters Sets

<table>
<thead>
<tr>
<th>Buffer 1</th>
<th>Knockout $K$</th>
<th>Sunshine $S$</th>
<th>Helical $H$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_1 = {4, 12}$</td>
<td>$S_1 = {3, 1, 12}$, $S_2 = {2, 2, 12}$, $S_3 = {4, 1, 12}$, $S_4 = {3, 2, 12}$, $S_5 = {3, 3, 12}$</td>
<td>$H_1 = {3, 3, 6}$</td>
<td></td>
</tr>
<tr>
<td>Buffer 2</td>
<td>$K_2 = {5, 20}$</td>
<td>$S_6 = {3, 2, 21}$, $S_7 = {4, 1, 20}$</td>
<td>$H_2 = {6, 6, 12}$</td>
</tr>
</tbody>
</table>

### 3.2.4 Simulations Performed

To obtain performance data about each of the three architectures, the models created in ProModel were simulated under different traffic types, each of which had their own set of parameters which were varied. Table 5 indicates what parameters sets where chosen to simulate for each of the
different traffic types. Only one specific group of values for the architecture configuration parameter sets and traffic type parameter sets could be simulated at a time. The choice of the which architecture configuration and traffic type parameter sets were being simulated is referred to as a simulation scenario. Each simulation scenario was simulated multiple times in order to collected enough sample points from the models to obtain meaningful performance data. Each individual execution of a simulation scenario is called a replication. Each replication was executed under a different random number seed so that no two replications for a simulation scenario ever executed identically due to the same cell arrival sequences. Table 6 indicates which simulation scenarios were performed. In general, all possible combinations of architecture configuration and traffic type parameter sets were simulated, except for the architecture configurations $S_3$, $S_4$, and $S_5$ which were only simulated with random traffic.

Table 5 Simulated Traffic Configurations Described with Parameters Sets

| Random $R$ | $R_1 = \{0.5\}, R_2 = \{0.6\}, R_3 = \{0.7\}, R_4 = \{0.75\}$  
|            | $R_5 = \{0.8\}, R_6 = \{0.85\}, R_7 = \{0.9\}, R_8 = \{1\}$       |
|            | $R_x = \{p\}$ where $p$ is the probability of a cell arrival       |
| CBR $C$    | $C_1 = \{10\}, C_2 = \{20\}, C_3 = \{50\}, C_4 = \{100\}$          |
| BurS $BU$  | $BU_1 = \{3,4\}, BU_2 = \{3,6\}, BU_3 = \{3,8\}, BU_4 = \{3,10\}$ |
|            | $BU_5 = \{5,4\}, BU_6 = \{5,6\}, BU_7 = \{5,8\}, BU_8 = \{5,10\}$ |
|            | $BU_9 = \{5,12\}, BU_{10} = \{5,14\}, BU_{12} = \{5,16\}$          |
|            | $BU_{13} = \{5,20\}, BU_{14} = \{7,10\}, BU_{15} = \{7,15\}$       |
|            | $BU_{16} = \{7,20\}, BU_{17} = \{7,25\}, BU_{18} = \{7,30\}$       |
|            | $BU_x = \{l, t\}$ where $l$ is the length of a burst and $t$ the interarrival time between bursts |

- 99 -
## Table 6: Simulation Scenarios Performed

<table>
<thead>
<tr>
<th></th>
<th>$K_1$</th>
<th>$K_2$</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
<th>$S_5$</th>
<th>$S_6$</th>
<th>$S_7$</th>
<th>$H_1$</th>
<th>$H_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$R_2$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$R_3$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$R_4$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$R_5$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$R_6$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$R_7$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$R_8$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$C_1$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$C_2$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$C_3$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$C_4$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$BU_1$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$BU_2$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$BU_3$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$BU_4$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$BU_5$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$BU_6$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$BU_7$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$BU_8$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$BU_9$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$BU_{10}$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$BU_{11}$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$BU_{12}$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$BU_{13}$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$BU_{14}$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$BU_{15}$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$BU_{16}$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$BU_{17}$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$BU_{18}$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

✓ = Scenario simulated  
† = Scenario not simulated  
✗ = Scenario not simulated; Beyond the capabilities of the simulator to measure
3.3 Performance Parameters Studied

To evaluate the performance of the architectures under the various types of traffic conditions, data was collected to generate several performance metrics relevant to ATM. Many of the parameters listed in literature are suffixed by the term rate. This is a misleading term because it implies that a time factor is present in the definition of the performance parameter it is associated with. All the measured and unmeasured parameters in the thesis will by referred to as probabilities if they are unitless and as rates if there is a time factor present in the definition of the associated parameter.

The first and most important parameter which was measured for each architecture under all simulations was cell loss probability (CLP), which is often called the cell loss ratio (CLR). This parameter measures the number of cells lost within the switching architecture for whatever reason compared to the total number of cells submitted to the switch for routing.

\[
\text{Cell Loss Probability (CLP)} = \frac{\text{Cells Lost}}{\text{Total Cells Submitted}} \quad \text{Eq. 35}
\]

Another critical factor in ATM switch design which was measured was the mean cell delay, which measures the average time (measured in time slot for this thesis) between a cell's arrival at a switch input port and its departure on the destined output port. This mean delay is composed of two components, a fixed delay (FD) and a queueing delay (QD) caused by cell buffering. When there is no load on a switching architecture, a single cell submitted to the switch will only experience the fixed delay. The mean cell delay does not include cells which were lost during switching. Only the time of cells which were successfully routed through the architecture are included in this calculation.

\[
\text{Mean Cell Delay (MCD)} = \frac{\sum_i (QD_i + FD)}{\text{Total Cells Successfully Routed}} \quad \text{Eq. 36}
\]

Another useful parameter is the throughput measurement. This parameter is directly related to the CLP and measures the ratio of successfully routed cells to the total number of cells submitted to the ATM switch.

\[
\text{Throughput} = \frac{\text{Total Cells Transmitted} - \text{Cells Lost}}{\text{Total Cells Transmitted}} = 1 - \text{CLP} \quad \text{Eq. 37}
\]

Another parameter which was measured in each of the architectures was the mean buffer utilization. This parameter is used to describe the average content level of all the buffers in a switching
architecture. The determination of this parameter varied depending on which architecture was being examined. Since the Knockout and Sunshine switch both use shared output buffers, the utilization of any one output buffer \( i \) could be determined by dividing the average number of cells it contained \( f_i \) by its cell capacity, \( B \). Summing all the utilizations for the \( N \) different buffers and dividing the result by \( N \) yields the MBU for the entire switch.

\[
\text{Mean Buffer Utilization (MBU)} = \frac{\sum_{i=1}^{N} \left( \frac{f_i}{B} \right)}{N} \quad \text{(for KOS & Sunshine)}
\]

Determining the MBU for the Helical architecture varies slightly because the buffers in the Helical architecture are distributed at each stage of the switch. First, the average buffer utilization at a stage \( n \) (where \( n = \log_2 N \)) is computed by the inner term shown in equation 39. The term \( f(\ell, n) \) denotes the average cell content of the \( \ell \)th buffer at stage \( n \), while \( B_n \) indicates the cell capacity of the buffers at stage \( n \). The average utilizations of the different stages are then averaged together to rendered the MBU for the entire switch.

\[
\text{Mean Buffer Utilization (MBU)} = \frac{\sum_{\ell=1}^{2N} \left( \frac{f(\ell, n)}{2N} \right)}{n} \quad \text{(for Helical)}
\]

3.4 Performance Parameters Not Studied

Several of the performance parameters associated with ATM switching systems were not measured in this simulation study for various reasons which will be explained. The first parameter not considered in this thesis was the Bit Error Rate (BER) mentioned in section 1.2.2.1. This parameter attempts to measure the rate at which single bit errors occur during physical transmission. This parameter was not considered or integrated into the performed simulations for two reasons. First, this error rate is more closely coupled with the B-ISDN physical layer than the ATM layer. Second, if the bit error occurs in the payload of the cell, it will be ignored by the switching system and must be detected by the end user. If the error occurs in a header bit of a cell, the HEC field of the cell header can be used by the input module of the switch to correct single bit errors. Thus, this type of error can be assumed to be transparent to the functionality and performance of the ATM layer and the switching architectures.
**Bit Error Rate (BER)**  
\[
\text{Bit Error Rate (BER)} = \frac{\text{Erred Bits}}{\text{Total Transmitted Bits}}
\]  
Eq. 40

Although infrequent, a possibility exists that bursts errors can occur during cell transmission. Burst errors are tightly grouped, single bit errors which may infect one cell or span several hundred cells. When the multiple errors occur in the cell's header, the errors may be detected but not fixed, and the erred cell must be discarded. The Cell Error Probability (CEP) measures the possibility of this occurrence by measuring the number of erred cells to the total number of cells submitted to the switching architecture.

\[
\text{Cell Error Probability (CEP)} = \frac{\text{Erred Cells}}{\text{Total Cells Submitted}}
\]  
Eq. 41

A small probability exists, however, that the HEC will fail to detect a cell with numerous bit errors in the header. In this case, the switching architecture will attempt to route the cell as though it were not flawed. Because of this, a cell can be unintentionally routed to the wrong destination, and that destination will accept the cell as a correct one. These cells are referred to as misinserted cells. The last major performance parameter not appraised during simulation was the Cell Misinsertion Rate (CMR) because it is dependent on the BER rate and cannot be controlled with the design of the ATM switch.

\[
\text{Cell Misinsertion Rate (CMR)} = \frac{\text{Misinserted Cells}}{\text{Time Interval}}
\]  
Eq. 42

### 3.5 Simulation Considerations

There were many considerations and issues which had to be addressed during the simulation of the architectural models. The most important factor was how to determine the number of replications of a particular simulation scenario that were needed to obtain data with statistical relevance. Another consideration was to determine if any limitations should be placed on the results collected from the simulation. The two following sections briefly detail some of these considerations. The final section describes some aspects of the overall simulation study performed.

#### 3.5.1 Simulation Replications
The most significant concern during simulation was determining how many replications of a particular simulation scenario were needed to ensure that the results which were being obtained were reliable and accurate within certain error limits. The CLP performance parameter was the critical factor in determining the number of replications needed because of the difficulty of measuring the number of cells discarded by the switching architecture. Only one cell may have been lost for every ten thousand, hundred thousand, or million cells submitted to the switch. Thus, obtaining an accurate average number of discarded cells was the primary concern. The other measured performance parameters like cell delay were measured on a per submitted cell basis for each replication, and therefore, a substantial amount of sample points were available to generate an average value for these replications. Because of the large number of sample points taken for these averages, the average values between replications exhibited low variability and required only a few replications to obtain final averages which were very precise. Determining the number of replications to simulate in order to obtain an accurate average of the number of cells discarded was influenced by other factors. These factors included the amount of time the simulation scenario was allowed to run (in other words, the total number of cells submitted to the switch), the magnitude of the number of cells which were discarded in each replication, the amount of error which could be tolerated in the averaged result, and the desired confidence interval of that result. Of these, the major factor which determined the number of replications needed was the magnitude of the number of cells discarded during the course of the replication, which is turn directly related to the quantity of cells submitted to the model under simulation. The more cells submitted, the more cells would be discarded.

Because of the rarity of discarded cells, it was possible that as few as one or two cells could be lost across fifty or more replications which when totalled together accounted for nearly $10^7$ submitted cells. Let the event of a cell being discarded be considered a random variable $x$. The average value of $x$ was a function of the number of replications, $n$, and is denoted by $x(n)$. Because of this infrequency of the event $x$, an average value of $x$ could be obtained from the collection of sample points in the $n$ replications, but when the collection of $x$ values were relatively small (numbers around 10 and lower) the average value of $x$ would have a large variance. Variance describes how much difference (variability) there is between an average value and the sample points it was computed from. Because of the large variance, the calculated confidence interval for the variable $x$ (the interval where the true average value of $x$ is assume to lie) would be extremely large when compared to the magnitude of $x(n)$, reducing the accuracy of the obtained results. When more cells were discarded lost during the simulation, the variance of the average value $x(n)$ was decreased,
and the calculated confidence interval was likewise decreased, suggesting a more precise result had been obtained.

The method used to determine if enough replications for a particular simulation scenario had been obtained was an iterative approach with two variations based on either an absolute or relative error. For the sake of discussion, absolute error will be used here. To start, an absolute error \( e \) was chosen for the average number of cells discarded, \( x(n) \), during the simulation, such as within \( \pm 2 \) cells. After this, an initial number of replications, \( n_0 \), were simulated to gather some initial sample points. The number of replications simulated initially varied from 3 or 4 up to 50, depending on the model, the configuration of that model, the traffic type, and the traffic parameters. From this data, the average number of discarded cells was computed along with the confidence interval based on the \( t \) distribution according to the equation

\[
\bar{x}(n) \pm (t_{n-1, 1 - \frac{\alpha}{2}}) \cdot \sqrt{\frac{s^2(n)}{n}}
\]

*Eq. 43*

where \( s^2 \) denotes the variance in the \( n \) sample points and \( \alpha \) the confidence level desired. For this thesis, the confidence level was chosen to be 90%. The confidence level specifies the probability that the true average of the number of discarded cells will fall into a range of numbers indicated by the confidence interval, where the true average is the average computed from infinite replications. Half of the confidence interval must therefore be smaller than the absolute error specified in order for the \( n \) replications to be sufficient.

\[
(\frac{t}{n-1, 1 - \frac{\alpha}{2}}) \cdot \sqrt{\frac{s^2(n)}{n}} \leq e
\]

*Eq. 44*

If the absolute error is smaller than the half of the confidence interval, then the confidence interval is too large and the \( n \) replications were not enough to compute an average number of cells lost within that specified error \( e \). More replications must be performed at this point to increase the reliability of the computed average. Overall, the total number of replications needed, \( n_a \), for an absolute error \( e \) can be expressed as

\[
n_a = \min \left\{ n \geq n_0 : \left(\frac{t}{n-1, 1 - \frac{\alpha}{2}}\right) \cdot \sqrt{\frac{s^2(n)}{n}} \leq e \right\}
\]

*Eq. 45*
In some simulation cases, this method was used not with an absolute error, but with a relative error \( r \) which indicated the error as a percentage \( r \) of the average number of discarded cells. The same sequence of steps was followed except that \( e \) was replaced according to equation 46.

\[
e = r \cdot \bar{x}(n)
\]

Eq. 46

The total number of replications needed, \( n_r \), for a relative error \( r \) can be expressed as

\[
n_r = \min \left\{ n \geq n_0 : \frac{( t \frac{1}{n-1}, 1 - \frac{r^2}{2} ) \cdot \sqrt{\frac{s^2(n)}{n}}}{|\bar{x}(n)|} \leq r \right\}
\]

Eq. 47

Both approaches were used to determine the number of replications needed. This was because each approach generated unreasonable replication estimates when the number of discarded cells approached extreme values like 0 and large numbers like 1000 and higher. For instance, if an absolute error of two cells was specified when approximately two thousand cells were being lost during a simulation, then the variability of the average number of cells lost would be large. This would in turn result in a large confidence interval not satisfying the absolute error. Hundreds or thousands of replications would have to be performed in order to reduce the variability of the computed average to satisfy the specified error requirement. The underlying problem is that the size of the error was disproportionate (when converted, the example absolute error is equivalent to a relative error of 0.1%) when compared to the total number of cells lost. Using a relative error (or increasing the absolute error) on the other hand eliminated this problem by making the error value proportional to the magnitude of the average number of discarded cells. This same situation applied in the reverse case when the number of cells being lost was very small. During simulation, a relative error rate of 10% was specified. In some instances though, this error rate required thousands of replications to be simulated since the number of cells being lost was relatively low. To overcome this, higher relative error rates were allowed for these cases and are reported in the analysis section.

**3.5.2 Limitations of Results**

As was briefly mentioned earlier in the thesis, to simulate cell loss probabilities on the scale of \( 10^6 \) for a switch architecture would require on the order of \( 10^6 \) to \( 10^{10} \) cells to be simulated through the switch. Simulating this number of cells would be possible given enough computing resources, time, and only a few architecture configurations. Since the simulator chosen in this thesis was only
capable of simulating approximately $10^6$ cells, the three different architectures were designed to perform at lower levels of performance so as be able to compare their relative performance under different conditions. Their CLP performance typically ranged from $10^{-1}$ to $10^{-6}$. Therefore, the CLPs generated from the simulations were valid down to values around $10^{-5}$. In a few simulation scenarios, the CLPs were of questionable reliability, but they tended to follow a general trend, therefore, they will be reported in the next chapter. Those questionably values will be clearly indicated in the results section of this thesis. In yet other cases, no CLP performance data could be collected for the simulated architecture because no cells were discarded in all the replications run, even when the replications were allowed to run longer periods of time. In addition, most models in a specific simulation scenario were simulated until the CLP was within approximately a 10% error range at a confidence level of 90%. These error ranges will be indicated in the results section also.
3.5.3 Length and Time of Simulation Runs

This section is provided to give the reader an estimate of the amount of time consumed by simulation and the length of time each replication was allowed to run. The first table in this section indicates the approximate time to simulate the three models for one random traffic replication for two quantities of submitted cells. The simulation time varied slightly depending on the loading factor used for the random traffic. Lower loading factors executed slightly faster than higher loading factors because less processing was required to handle discarded cells since fewer cells were being lost. The computer platform used had a much larger impact. The values listed in table 7 are for simulations performed on a 120MHz Pentium. Typically, slower computers were used. The average computer configuration was a Windows NT, Pentium workstation running at 90 MHz, and the simulation times for a simulation scenario run on this system were about 25 to 30% longer than those for the 120MHz system. In some cases, 486 based machines were utilized, which performed the same simulations in about 2 to 3 times the amount of time listed in table 7.

<table>
<thead>
<tr>
<th>Model</th>
<th>$10^5$ Cells</th>
<th>$10^6$ Cells</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knockout</td>
<td>3 min</td>
<td>30 min</td>
</tr>
<tr>
<td>Sunshine</td>
<td>3.5 min</td>
<td>36 min</td>
</tr>
<tr>
<td>Helical</td>
<td>15 min</td>
<td>2.5 hrs</td>
</tr>
</tbody>
</table>

The following table relates some of the logistic figures about the simulation performed to in order to collect data needed to compare the three architectures.

<table>
<thead>
<tr>
<th>Statistics on Performed Simulations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Scenarios Simulated = 215</td>
</tr>
<tr>
<td>Total Replications Simulated = 8500</td>
</tr>
<tr>
<td>Total Hours of Simulation (approx) = 3000 hrs</td>
</tr>
</tbody>
</table>
4 Results and Analysis

This section examines the results of the simulations which were performed according to section 3. The architectures will be compared according to the simulated traffic types.

4.1 Random Traffic Simulation Results

The following section will provide graphical analysis of the relative performance of the three architectures under the different architecture configurations and varying loads of random traffic. First, general conclusions about the data portrayed in the graphs will be presented. After these conclusions, the graphical data will follow. Finally, some of the conclusions listed below will be further explained at the end of this section.

4.1.1 Conclusions

1. For buffer 1 configurations, \( K_1 = \{4, 12\} \) and \( S_1 = \{3,1,12\} \) CLP performance was identical while \( S_2 = \{2,2,12\} \) performed slightly worse for lower traffic loads. The Helical switch \( H_1 = \{3,3,6\} \) exhibited much higher CLP for higher traffic loads, but below 80% to 85%, its CLP decreased at a rate much greater than \( K_1, S_1, \) and \( S_2 \). Below a 50% load, Helical \( H_1 \) CLP performance was 1 order of magnitude better than the other architectures. (Figure 46)

2. For buffer 2 configurations, no noticeable differences were detected between the CLP performance of \( K_2 = \{5,20\}, S_2 = \{3,2,21\}, \) and \( S_2 = \{4,1,20\} \) within error considerations, but CLP performance is approximately one order of magnitude greater than the buffer 1 configurations. The Helical switch again performed much worse than the other three switch configurations for high random traffic loads, but below the 85% input load, it demonstrated a significant rate of decreasing CLP. At a 70% load, \( H_2 \) CLP performance was 2 orders of magnitude better than the \( H_1 \) configuration shown in figure 46. (Figure 47)

3. For input loads below 80%, Sunshine buffer 1 architectures with higher values for \( k+T \) \((S_3=\{4,1,12\} \) and \( S_2 = \{3,2,12\}\)) demonstrated a higher rate of CLP decrease than those with lower values of \( k+T, S_1 = \{3,1,12\} \) and \( S_2 = \{2,2,12\}\). Generally, the improvement in performance is due to the increased number of Banyan routing networks, \( k \). (Figure 48)

4. For the Knockout and all Sunshine buffer 1 configurations, the average cell delay time started slightly above two time slots (approximately the fixed delay) and increased with increasing input loads to a maximum of 7 time slots. The Helical switch demonstrated an elevated delay time starting around 4.5 time slots, which increased to 12 time slots under full load conditions. (Figure 49)

5. Buffer 2 configurations exhibited similar delay trends as buffer 1 configurations, only slightly higher. The maximum delay of the Helical switch \( H_2 \) increased dramatically to 24 time slots
under the full load condition compared to approximately 12 time slots for \( K_2, S_6, \) and \( S_7 \) architecture configurations. (Figure 50)

6. Utilization of output buffers for buffer 1 and 2 architecture configurations increased gradually from less than 10% at 50% input loads to a maximum of 50% utilization under full load conditions. For all input loads, the Helical switch exhibited overall buffer utilization rates approximately 10% to 20% higher than the other architectures. (Figures 51 & 52)

7. For lower input loads, cells were lost primarily at the Knockout concentrator, the Sunshine recirculator, and the second and third Helical buffer stages. As input loads increased above 70%, cell lost shifted to the Knockout and Sunshine switch output buffers and the first buffer stage of the Helical switch. (Figures 55 & 56)

8. As eluded to by conclusion 7, the first buffer stage of the Helical switch became a bottleneck to performance, limiting the number of cells delivered to the following two stages. This can be seen in the buffer utilizations per stage as the input load increases. Stage 1 buffers approached 100% utilization while buffers at stages two and three ranged between 50% and 70% utilization. (Figures 53, 54, 55, & 56)

9. For input loads below 80%, the Knockout, Sunshine, and Helical architectures achieved a throughput approaching 1. Under full load conditions, the Knockout and Sunshine architectures maintained throughputs above 95% while the Helical switch only maintained an 82% to 83% throughput. (Table 9 & 10)

![Graph](image-url)  
**Figure 46** CLP of \( K_1, S_1, S_2, \) and \( H \), Buffer 1 Architectures under Random Traffic

- 110 -
Figure 47  CLP of $K_2$, $S_6$, $S_7$, and $H_2$ Buffer 2 Architectures under Random Traffic
**Figure 48** CLP of Sunshine Configurations $S_1$, $S_2$, $S_3$, $S_4$, and $S_5$ under Random Traffic
Figure 49  Cell Delay of $K_1$, $S_1$, $S_5$, and $H_1$ Architectures under Random Traffic
Figure 50  Cell Delay of $K_2$, $S_6$, $S_7$, and $H_2$ Architectures under Random Traffic
Figure 51 Buffer Utilization of $K_1$, $S_1$, $S_2$, and $H_2$ Architectures under Random Traffic
Figure 52 Buffer Utilization of $K_2$, $S_6$, $S_7$, and $H_2$ Architectures under Random Traffic
Figure 53 Buffer Utilization at each Stage of the Helical $H_i$ Architecture under Random Traffic
Figure 54 Buffer Utilization at each Stage of the Helical H₂ Architecture under Random Traffic
Figure 55 Locations and Quantities of Cell Loss in Buffer 1 Architectures under Random Traffic loads 50%, 60%, and 70%
Figure 56 Locations and Quantities of Cell Loss in Buffer 1 Architectures under Random Traffic loads 80%, 90%, and 100%

CCL = Concentrator Cell Loss
RCL = Recirculator Cell Loss
BCL = Buffer Cell Loss
### Table 9 Throughput Ranges of Buffer 1 Architectures under Random Traffic

<table>
<thead>
<tr>
<th>Load</th>
<th>Knockout K₁</th>
<th>Sunshine S₁</th>
<th>Sunshine S₂</th>
<th>Helical H₁</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Low</td>
<td>High</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>50</td>
<td>0.999879</td>
<td>0.999901</td>
<td>0.999883</td>
<td>0.999904</td>
</tr>
<tr>
<td>60</td>
<td>0.999773</td>
<td>0.999805</td>
<td>0.999783</td>
<td>0.999819</td>
</tr>
<tr>
<td>70</td>
<td>0.999541</td>
<td>0.999577</td>
<td>0.999539</td>
<td>0.999594</td>
</tr>
<tr>
<td>80</td>
<td>0.998392</td>
<td>0.998514</td>
<td>0.998374</td>
<td>0.998554</td>
</tr>
<tr>
<td>90</td>
<td>0.990961</td>
<td>0.991270</td>
<td>0.990935</td>
<td>0.991298</td>
</tr>
<tr>
<td>100</td>
<td>0.960615</td>
<td>0.961057</td>
<td>0.960559</td>
<td>0.961403</td>
</tr>
</tbody>
</table>

### Table 10 Throughput Ranges of Buffer 2 Architectures under Random Traffic

<table>
<thead>
<tr>
<th>Load</th>
<th>Knockout K₂</th>
<th>Sunshine S₆</th>
<th>Sunshine S₇</th>
<th>Helical H₂</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Low</td>
<td>High</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>70</td>
<td>0.999983</td>
<td>0.999985</td>
<td>0.999979</td>
<td>0.999983</td>
</tr>
<tr>
<td>75</td>
<td>0.999972</td>
<td>0.999976</td>
<td>0.999969</td>
<td>0.999973</td>
</tr>
<tr>
<td>80</td>
<td>0.999942</td>
<td>0.999953</td>
<td>0.999943</td>
<td>0.999952</td>
</tr>
<tr>
<td>85</td>
<td>0.999785</td>
<td>0.999813</td>
<td>0.999804</td>
<td>0.999857</td>
</tr>
<tr>
<td>90</td>
<td>0.998706</td>
<td>0.998856</td>
<td>0.998991</td>
<td>0.999119</td>
</tr>
<tr>
<td>100</td>
<td>0.977107</td>
<td>0.977368</td>
<td>0.978153</td>
<td>0.978625</td>
</tr>
</tbody>
</table>

### 4.1.2 Explanations

As can be clearly be seen from the graphs, the Helical switch CLP performance was much worse than the other two architectures for loads above 50% to 60%. The primary reason for this was mentioned in conclusion 8; the first stage buffer becomes a bottleneck. Under the full load condition, each of the \( N \) input ports will receive and process one real cell, passing it onto the broadcast units. Since the cells arriving at the broadcast units are all real, each unit will create two cells, a real and dummy version of the original cell. The broadcast units will then forward these cells to the first stage buffers, one cell to each buffer. Thus, during each time slot, a cell will arrive at each buffer in the first stage. Since incoming cells are assigned output addresses according to a uniform, random distribution, it is possible that more than \( N/2 \) real cells could arrive in a given time slot at the buffers servicing a first stage concentrator. Assume for the moment that \( k \) real cells
arrive at the first stage buffers servicing a particular first stage concentrator, where \( k > N/2 \). Since the first stage concentrator can only process \( N/2 \) real cells, \( k - N/2 \) real cells will be forced to remain in their HOL buffer position until the next time slot when they can be processed. In the next time slot each of these buffers will receive another cell, and the buffers which contain real cells which were deferred by one time slot will have to hold the newly arriving cells one time slot while the deferred HOL cells are processed. Each time more than \( N/2 \) real cells arrive at the first stage buffers supplying a particular concentrator, the combined buffer space is reduced by \( k - N/2 \). Eventually, the first stage buffers fill to capacity, and cells are dropped. This process is clearly illustrated in figures 53, 54, and 56. In figure 56, as the input load increases, the number of cells lost at the first stage buffer increases from being the least significant contributor of the three buffer stages at an 80% load to being the most significant contributor of lost cells at a 100% load. Figures 53 and 54 illustrate that the first stage buffers of the \( H_1 \) configuration reach almost 90% utilization while the first stage buffers of the \( H_2 \) configuration approach 100% buffer utilization, both of which confirm this conclusion. The basic reason for this high buffer utilization stems from the fact that a dummy cell is created for each real cell submitted to the next stage of the switch. Thus, for an \( N=8 \) switch, there are 3 stages, and a total of 3 dummy cells are created for each real cell successfully routed through the switch. Each of these dummy cells occupies buffer space, increasing buffer utilization.

Conclusion 1 also needs further explanation. A slight CLP performance gap was experienced between Sunshine configurations \( S_1 \) and \( S_2 \). This gap is primarily due to the number of Banyan networks present in the two configurations. In the \( S_1 \) configuration, 3 parallel Banyan networks were present while in the \( S_2 \) only two are present. As the input load increases from 50%, the probability that out of all the arriving cells in a given time slot there are three cells destined to the same output port also increases. The \( S_1 \) configuration can route all three cells simultaneously to the output buffers, while the \( S_2 \) configuration can only route two of the three cells to the output buffers. One cell must be recirculated. If in the next time slot, three or more cells arrive destined to the same output port that the recirculated cell is destined for, the chance of cells being dropped at the recirculator increases very quickly. This is demonstrated in figure 55. More cells are lost at the recirculator in the \( S_2 \) configuration for loads between 50% and 70% than in the \( S_1 \) configuration. This is because the reduced number of Banyan networks in the \( S_2 \) configuration forces more cells to be recirculated, which naturally increases the amount of cells lost at the recirculator for a fixed recirculator cell loss probability. As the input load increases above 70% (figure 56), the magnitude of cells lost at the output buffers become more significant than those lost at the recirculator, and the CLP performances of the two configurations converge. As a consequence of this situation, because
more cells are being lost in the $S_2$ configuration, fewer cells are being delivered overall to the output buffers. This reduces the average content of the output buffers, which in turn, suggests that the $S_2$ configuration should have a slightly smaller delay time than the $S_1$ configuration. This conclusion is confirmed in figure 49.

Given the gap of CLP performance experienced between the $S_1$ and $S_2$ configurations, it would be expected that the same gap would be experienced between the $S_6$ and $S_7$ configurations. As figure 48 indicates, this was not the case. In fact, there is virtually no difference, within error limitations, between the CLP performance of the $S_6$ and $S_7$ configurations. The most likely explanation for the missing gap is demonstrated in figure 32 back in section 2.2.3. Even though this figure depicts the recirculator CLP for a 100% load, the trend is similar for lower load factors. The figure indicates that as the number of Banyan routing networks is increased, a diminishing improvement in recirculator CLP performance is achieved. This is because the probability of there being more than $k$ cells destined to the same output port in a given time slot decreases as $k$ increases. Thus, fewer cells recirculate and risk being discarded by the recirculator in the $S_6$ and $S_7$ configurations. The small number of cells that do recirculate can easily be managed by the number of recirculator lines chosen. The little difference that does exist between $S_6$ and $S_7$, if not due to error, could also be attributed to the fact that $S_6$ has one more buffer per output port than does $S_7$ for reasons explained in section 3.2.3.

4.2 CBR Traffic Simulation Results

The following section will provide both graphical and tabular analysis of the performance of the three architectures under the different architecture configurations and different CBR traffic inputs. First, general conclusions about the data portrayed in the graphs and tables will be presented. After these conclusions, the graphical and tabular data will be presented. Finally, some of the conclusions will be further explained at the end of the section.

As described in section 3.2.2.2, two versions of CBR traffic were simulated. In one version, the distribution of cells within the frame structure was computed at the beginning of each frame. In the second version, the same distribution was only computed once at the beginning of each simulation, and that particular frame was continually repeated. The following two sections will separate the results according to this distinction.
### 4.2.1 Nonrepeated Frame

#### 4.2.1.1 Conclusions

1. For buffer 1 configurations, \( K_1 = \{4, 12\} \) and \( S_1 = \{3,1,12\} \) exhibited identical CLP performance while \( S_2 = \{2,2,12\} \) performed slightly worse for small frame sizes. The Helical switch \( H_1 = \{3,3,6\} \) exhibited a constant CLP of less than one. As frame sizes increased, the CLP performance of each switch converged to the performance measured under a 100\% random traffic load. For small frame sizes, CLP performance for \( K_1, S_1, \) and \( S_2 \) was 1 to 2 orders of magnitude better than under random traffic. (Figure 57)

2. For buffer 2 configurations, Helical switch \( H_2 \) CLP was identical to the \( H_1 \) CLP and was several orders of magnitude worse than CLP performance of the other two architectures. No discernable differences existed between the CLP performance of \( K_2 = \{5,20\}, S_7 = \{4,1,20\}, \) but the Sunshine switch \( S_6 = \{3,3,21\} \) demonstrated a slight CLP performance increase over \( S_7 \) and \( K_2 \). (Figure 58)

3. For the Knockout and all Sunshine buffer 1 configurations, the average cell delay time was approximately 7 time slots, with a slight reduction for small frame sizes. \( S_2 \) routed cells approximately 3 to 4% faster than \( S_6 \) and \( K_2 \). The Helical switch cell delay was identical to the random traffic case, 12 time slots for all frame sizes. (Figure 59)

4. Buffer 2 configurations exhibited similar cell delay times as buffer 1 configurations, only higher. The maximum delay of the Helical switch \( H_2 \) was 24 time slots for all frame sizes while only 12 for \( K_2, S_6, \) and \( S_7 \). Sunshine \( S_6 \) was approximately 3\% faster than \( K_2 \) and \( S_7 \) (Figure 60)

5. Buffer utilization for Knockout and Sunshine buffer 1 architectures was constant at 50\% except for a frame size of 10, which was around 43\%. Buffer utilizations for these architectures dropped 20\% and 10\% for frame sizes of 10 and 20, respectively, when changed to buffer 2 configurations. The Helical buffer utilization declined slightly from 80\% to 77\% for both buffer configurations as frame sizes increased. (Figures 61 & 62)

6. All architectures for both buffer configurations demonstrated a 1\% to 2\% improvement in throughput performance for large frame sizes (F=100) when compared to full load, random traffic throughput performance, except for the Helical switch whose throughput increased marginally by less 0.5\%. (Tables 11 & 12)

#### Table 11 Throughput Ranges for Buffer 1 Architectures under CBR, Nonrepeated Traffic

<table>
<thead>
<tr>
<th>Frame Size (cells)</th>
<th>Knockout ( K_1 ) Low</th>
<th>Knockout ( K_1 ) High</th>
<th>Sunshine ( S_1 ) Low</th>
<th>Sunshine ( S_1 ) High</th>
<th>Sunshine ( S_2 ) Low</th>
<th>Sunshine ( S_2 ) High</th>
<th>Helical ( H_1 ) Low</th>
<th>Helical ( H_1 ) High</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.999617</td>
<td>0.999648</td>
<td>0.999592</td>
<td>0.999619</td>
<td>0.998269</td>
<td>0.998326</td>
<td>0.843435</td>
<td>0.843679</td>
</tr>
<tr>
<td>20</td>
<td>0.997733</td>
<td>0.997862</td>
<td>0.997818</td>
<td>0.997916</td>
<td>0.995591</td>
<td>0.995759</td>
<td>0.837934</td>
<td>0.838178</td>
</tr>
<tr>
<td>50</td>
<td>0.985194</td>
<td>0.985556</td>
<td>0.985065</td>
<td>0.985458</td>
<td>0.983570</td>
<td>0.983963</td>
<td>0.832619</td>
<td>0.833120</td>
</tr>
<tr>
<td>100</td>
<td>0.972815</td>
<td>0.973546</td>
<td>0.972186</td>
<td>0.972885</td>
<td>0.971655</td>
<td>0.972464</td>
<td>0.828146</td>
<td>0.828662</td>
</tr>
</tbody>
</table>
Table 12  Throughput Ranges for Buffer 2 Architectures under CBR, Nonrepeated Traffic

<table>
<thead>
<tr>
<th>Frame Size (cells)</th>
<th>Knockout $K_x$</th>
<th>Sunshine $S_y$</th>
<th>Sunshine $S_y$</th>
<th>Helical $H_z$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Low</td>
<td>High</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>10</td>
<td>0.999988</td>
<td>0.999991</td>
<td>0.999989</td>
<td>0.999991</td>
</tr>
<tr>
<td>20</td>
<td>0.999958</td>
<td>0.999968</td>
<td>0.999966</td>
<td>0.999971</td>
</tr>
<tr>
<td>50</td>
<td>0.999061</td>
<td>0.999149</td>
<td>0.999363</td>
<td>0.999441</td>
</tr>
<tr>
<td>100</td>
<td>0.993795</td>
<td>0.994172</td>
<td>0.995214</td>
<td>0.995562</td>
</tr>
</tbody>
</table>
Figure 57  CLP of $K_1$, $S_1$, $S_2$, and $H_1$ Architectures under Nonrepeated CBR Traffic
Figure 58  CLP of $K_2$, $S_6$, $S_7$, and $H_2$ Architectures under Nonrepeated CBR Traffic
Figure 59 Cell Delay of $K_1$, $S_1$, $S_2$, and $H_1$ Architectures under Nonrepeated CBR Traffic
Figure 60 Cell Delay of $K_2$, $S_6$, $S_7$, and $H_2$ Architectures under Nonrepeated CBR Traffic
**Figure 61** Buffer Utilization of $K_1$, $S_1$, $S_2$, and $H$, Architectures under Nonrepeated CBR Traffic
Figure 62 Buffer Utilization of $K_2$, $S_6$, $S_7$, and $H_2$ Architectures under Nonrepeated CBR Traffic
4.2.1.2 Explanations

The performance of the Helical switch under nonrepeated CBR traffic as it was implemented was very poor. It again demonstrated the limited throughput performance of approximately 83% with delay times varying between 12 and 24 time slots depending on the amount of the buffers incorporated into the switch. The reason for the Helical switches performance was described earlier in section 4.1.2 under the random traffic analysis.

Sunshine configuration $S_2$ CLP varied from the Knockout $K_r$ and Sunshine $S_i$ configurations because of its reduced number of Banyan networks, as mentioned in section 4.1.2. The reduced number of Banyan networks forced more cells to be recirculated, which increased the number of cells lost there. Because fewer cells were being routed directly to the buffers, $S_2$ demonstrated a minor reduction in delay time and buffer utilization compared $S_i$ and $K_r$.

4.2.2 Repeated Frame

Due to the nature of this traffic type, CLP could not be calculated because of the variability in the number of cells lost. In some simulation replications, the distribution of cells in the frame structure would inherently cause cell loss within the different architectures. Since the same cell pattern would be input into the architecture multiple times, the same number of cells would be lost for each submitted frame. The sum of all the cells lost for the entire replication would be of a large magnitude. In other frame distributions, no cells would be lost when the frame was submitted, so for the entire replication, no cells would be lost. The average number of cells lost based on multiple replications could be computed, but due to the described situation, the variability of the number of cells lost data was too large for reliable CLP calculations. To overcome this problem, identical frame patterns were submitted to each architecture, and thus the total number of cells lost and where they were lost, could be compared accurately between architectures. To provide perspective to the number of cells lost, the following number of total cells submitted to each architecture for the simulated frame sizes are

<table>
<thead>
<tr>
<th>Frame Size (cells)</th>
<th>10</th>
<th>20</th>
<th>50</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cells Submitted</td>
<td>$4\times10^5$</td>
<td>$8\times10^5$</td>
<td>$2\times10^6$</td>
<td>$4\times10^6$</td>
</tr>
</tbody>
</table>
listed in table 13. This particular traffic type provided only limited information about three architectures.
4.2.2.1 Conclusions

1. For all frame sizes, the Sunshine buffer 1 configuration $S_1$ with the most parallel Banyan routing networks (3) lost the same or fewer cells (RCL + BCL) than the Knockout configuration $K_1$ (CCL + BCL) and the Sunshine $S_2$ configuration. When the number of recirculator lines in $S_1$ and $S_2$ was increased to 4, configuration $S_2$ with only 2 Banyan routing networks discarded fewer cells for frame sizes above 20 cells than $S_1$ or $K_1$. Helical switch lost between 10 and 100 times more cells than the other two architectures. (Figure 63).

2. A similar trend as was stated in conclusion 1 was true for buffer 2 configurations. (Figure 64) Sunshine configuration $S_7$ with 4 Banyan routing networks lost the same or fewer cells when compared to the $K_1$, $S_6$, and $H_2$ configurations. When the number of recirculator lines was increased to 4 for $S_6$ and $S_7$, the $S_6$ configuration with only 3 Banyan networks lost fewer cells than the $S_7$ configuration for a frame size of 100. Below this, neither configuration lost any cells.

3. The cell delay for buffer 1 Sunshine and Knockout switches was nearly identical, ranging from 3.2 time slots for small frame sizes to 7.5 for a frame size of 100 cells. The Helical switch demonstrated similar performance to the Nonrepeated CBR traffic case. Sunshine configuration $S_2$ had a slightly lower delay due to fact that it lost more cells, reducing the average cell content in its output buffers and thus the average delay time. (Figure 65)

4. Cell Delay times for buffer 2 Sunshine and Knockout switches ($K_2$, $S_6$, and $S_7$) were almost identical to those of the buffer 1 configurations, except for about a 0.8 time slot difference for a frame size of 100 cells. The Helical buffer 2 configuration $H_2$ delay increased to approximately 24 time slots. (Figure 66)

The following conclusions were obtained for the Sunshine architecture by varying the number of recirculating lines $T$ from 1 to 4 for the configurations $S_1$, $S_2$, $S_6$, and $S_7$ during simulation.

5. For the Sunshine Architecture, as the number of parallel Banyan routing networks increases, the number of cells recirculated decreases. With 4 Banyan routing networks, only 0.12% of the submitted cells were recirculated compared to 9% to 12% for 2 Banyan networks. (Tables 14 - 16)

6. For the Sunshine Architecture, as the number of parallel Banyan routing networks increased, the need for more recirculating lines, $T$, was reduced since it was less likely that cells would be recirculated. (Tables 14 - 16)

7. For the Sunshine Architecture, as the number of parallel Banyan routing networks and recirculating lines increased, the full traffic load was effectively transferred to the output cell buffers, and more cells were lost there. (Tables 14 - 16)

8. When 3 or more Banyan routing networks were used in the Sunshine Architecture, increasing the number of recirculating lines above 2 achieved little performance gain. (Tables 14 - 16)
As the number of Banyan routing networks increases towards the number of concentrator output lines $L$ in the Knockout switch, the Sunshine architectures CLP performance approaches the performance of the Knockout switch. (Tables 14 - 16)
Figure 63  Cells Lost for $K_1$, $S_1$, $S_2$, and $H_1$ under Repeated CBR Traffic, Frame = 100 cells
Figure 64 Cells Lost for K₂, S₆, S₇, and H₂ under Repeated CBR Traffic, Frame = 100 cells
Figure 65 Cell Delay for $K_1$, $S_1$, $S_2$, and $H_1$ Architectures under Repeated CBR Traffic
Figure 66 Cell Delay of $K_2$, $S_6$, $S_7$, and $H_2$ Architectures under Repeated CBR Traffic
Table 14  Cells Recirculated for Various Values of \( k \) and \( T \) in the Sunshine Architecture under CBR, Repeated Traffic, Frame = 50 cells

<table>
<thead>
<tr>
<th>( k ) Banyan Networks</th>
<th>Recirculator Lines, ( T )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>186482</td>
</tr>
<tr>
<td>3</td>
<td>25000</td>
</tr>
<tr>
<td>4</td>
<td>2450</td>
</tr>
</tbody>
</table>

Table 15  Cells Lost at the Recirculator for Various Values of \( k \) and \( T \) in the Sunshine Architecture under CBR, Repeated Traffic, Frame = 50 cells

<table>
<thead>
<tr>
<th>( k ) Banyan Networks</th>
<th>Recirculator Lines, ( T )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>46541</td>
</tr>
<tr>
<td>3</td>
<td>2550</td>
</tr>
<tr>
<td>4</td>
<td>150</td>
</tr>
</tbody>
</table>

Table 16  Cells Lost at Buffers for Various Values of \( k \) and \( T \) in the Sunshine Architecture under CBR, Repeated Traffic, Frame = 50 cells

<table>
<thead>
<tr>
<th>Banyan Networks</th>
<th>Recirculator Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>994</td>
</tr>
<tr>
<td>3</td>
<td>3275</td>
</tr>
</tbody>
</table>

4.3  Bursty Traffic Simulation Results
This section presents bursty traffic simulation results and conclusions in graphical and tabular form for the three architectures under the different architecture configurations and varying bursty traffic parameters. First, general conclusions about the data portrayed in the graphs will be presented. After these conclusions, the graphical data will be presented. Finally, some of the conclusion will be further explained at the end of this section.

4.3.1 Conclusions

1. In general, as burst length increased, an architecture's CLP increased. As interarrival times increased, however, an architecture's CLP decreased. (Figures 67 - 72)

2. For buffer 1 configurations, \( K_1 \) and \( S_2 \), CLP performance was nearly identical, while \( S_2 \) (possessing fewer Banyan networks than \( S_1 \)) CLP performance diverged away from the CLP performance of \( K_1 \) and \( S_1 \) for the shortest mean burst length of 3 cells, becoming increasingly worse as the interarrival time increased. For a burst length of 3 and interarrival time of 12, \( S_1 \) had a CLP 55% better than the CLP of \( S_2 \). The CLP performance of the Helical switch \( H \) configuration surpassed all the configurations of the other two architectures. (Figure 67)

3. For large burst lengths, nearly identical CLP performances were recorded for the Knockout and Sunshine configurations \( K_1 \), \( S_1 \), and \( S_2 \), except \( S_6 \) performed a marginal 5 to 10% better than \( S_6 \) for a burst length of 7 and interarrival time of 30 and at other sample points. Again, the Helical switch achieved an increasingly lower CLP than the other configurations for increasing interarrival time. (Figures 68 & 69)

4. For buffer 1 configurations, CLP performance ranged from below \( 10^{-1} \) to \( 10^{-4} \) for the simulated traffic conditions. (Figures 67 - 69)

5. For buffer 2 configurations, the CLP performance trends of the three architectures were the same as the buffer 1 configurations. The Helical switch out performed the other two architectures by nearly one order of magnitude of CLP for larger interarrival times. The performance of the Sunshine and Knockout switch configurations was very similar, with \( S_2 \) performing one order of magnitude better than \( S_6 \) for smaller length bursts and long interarrival time. For larger bursts, \( S_6 \) CLP performance was slightly better (15-25%) than that for \( S_2 \) as interarrival time increased. (Figures 70 - 72)

6. For buffer 2 configurations, CLP performance ranged between \( 10^{-2} \) to \( 10^{-6} \) for the simulated traffic, an order of magnitude decrease from buffer 1 configurations of the same models. (Figures 70 - 72)

7. Knockout and Sunshine buffer 1 architecture configurations demonstrated identical cell delay times, which was expected because of their use of indentical buffering schemes. The cell delay varied from 3.5 to 6.5 time slots, with longer delays experienced under longer burst lengths. The Helical switch delays varied between 4.5 and 7.5 time slots, a one time slot difference from the Knockout and Sunshine architectures. In general, cell delay times were lower than those experienced in random traffic. (Figures 73 - 75)
8. Buffer 2 architecture configurations exhibited similar delay trends, only starting at larger values for smaller interarrival times and approaching the same delay time as buffer 1 configurations for large interarrival times. Again, the cell delays for the Knockout and Sunshine architectures were the same while the cell delay for the Helical switch was on average one to two time slots higher. (Figure 76)

9. The Knockout and Sunshine switches demonstrated identical buffer utilization, therefore, only Knockout switch data is shown. Overall buffer utilization was reduced dramatically compared to the other two traffic types for equivalent CLP performance, indicating the severe impact of bursty traffic on switching performance. (Figures 79 & 81)

9. Unlike the random traffic case, buffer stages 2 and 3 appear to become the bottlenecks in the performance of the Helical under bursty traffic, which show higher buffer utilizations than the stage 1 buffers. This directly translates into more cells lost at buffers at stages 2 and 3. (Figure 81)

10. The overall buffer utilization of the Helical switch H, configuration was only 1% to 5% higher than the Knockout and Sunshine architectures buffer utilizations for all bursty traffic conditions, compared to roughly a 20% difference in the random traffic cases. The same trend was seen in buffer 2 architecture configurations. (Figures 77 - 80)

11. For the shortest interarrival times, the throughput of each of the switches were very similar in both buffer 1 and 2 configurations. Throughput was above 95% for all cases, including the Helical switch which only had a throughput of 83% in the 100% load condition in random traffic. (Tables 17 & 18)
Mean Interarrival Time (time slots)

Mean Burst Length = 3

- Knockout K1
- Sunshine S1
- Sunshine S2
- Helical H1

Figure 67  CLP of K₁, S₁, S₂, and H₁ Architectures under Bursty Traffic (Length = 3)
Figure 68  CLP of $K_1$, $S_1$, $S_2$, and $H_1$ Architectures under Bursty Traffic (Length = 5)
Figure 69 CLP of $K_1$, $S_1$, $S_2$, and $H_1$ Architectures under Bursty Traffic (Length = 7)
Mean Interarrival Time (time slots)

Mean Burst Length = 3

- Knockout K2
- - - Sunshine S6
- ----- Sunshine S7
- - - Helical H2

Figure 70 CLP of K2, S6, S7, and H2 Architectures under Bursty Traffic (Length = 3)
Mean Interarrival Time

Mean Burst Length = 5

- - - Knockout K2
- - - Sunshine S6
- - - Sunshine S7
- - - Helical H2

Figure 71 CLP of K_2, S_6, S_7, and H_2 Architectures under Bursty Traffic (Length = 5)
Figure 72 CLP of $K_2$, $S_6$, $S_7$, and $H_2$ Architectures under Bursty Traffic (Length = 7)
Figure 73  Cell Delay of $K_1$, $S_1$, $S_2$, and $H_1$ Architectures under Bursty Traffic (Length = 3)
Figure 74  Cell Delay of $K_1$, $S_1$, $S_2$, and $H_1$ Architectures under Bursty Traffic (Length = 5)
Figure 75  Cell Delay of $K_1$, $S_1$, $S_2$, and $H_1$ Architectures under Bursty Traffic (Length = 7)
Figure 76  Cell Delay of $K_2$, $S_6$, $S_7$, and $H_2$ Architectures under Bursty Traffic (Length = 5)
Figure 77 Buffer Utilization of K1 Architecture under Bursty Traffic (Lengths = 3, 5, 7)
Figure 78 Buffer Utilization of H, Architecture under Bursty Traffic (Lengths = 3, 5, 7)
Figure 79 Buffer Utilization of $K_2$ Architecture under Bursty Traffic ($Lengths = 3, 5, 7$)
Figure 80 Buffer Utilization of $H_2$ Architecture under Bursty Traffic (Lengths = 3, 5, 7)
Burst Lengths

- Length = 3, H1 Buffer B1
- - - Length = 3, H1 Buffer B2
- - - - Length = 3, H1 Buffer B3
- - - - - Length = 5, H1 Buffer B1
- - - - - - Length = 5, H1 Buffer B2
- - - - - - - Length = 5, H1 Buffer B3

Figure 81 Buffer Utilization at each Stage of the H, Architecture under Bursty Traffic
Table 17 Worst Case Simulated Throughput Ranges for Buffer 1 Architecture Configurations under Bursty Traffic

<table>
<thead>
<tr>
<th>Burst Length</th>
<th>Knockout K₁</th>
<th>Sunshine S₁</th>
<th>Sunshine S₂</th>
<th>Helical H₁</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Low</td>
<td>High</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>3</td>
<td>0.993621</td>
<td>0.993841</td>
<td>0.993879</td>
<td>0.994419</td>
</tr>
<tr>
<td>5</td>
<td>0.962772</td>
<td>0.961977</td>
<td>0.961009</td>
<td>0.962909</td>
</tr>
<tr>
<td>7</td>
<td>0.975472</td>
<td>0.976557</td>
<td>0.975588</td>
<td>0.976786</td>
</tr>
</tbody>
</table>

Table 18 Worst Case Simulated Throughput Ranges for Buffer 2 Architecture Configurations under Bursty Traffic

<table>
<thead>
<tr>
<th>Burst Length</th>
<th>Knockout K₂</th>
<th>Sunshine S₆</th>
<th>Sunshine S₇</th>
<th>Helical H₂</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Low</td>
<td>High</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>3</td>
<td>0.999585</td>
<td>0.999885</td>
<td>0.999656</td>
<td>0.999768</td>
</tr>
<tr>
<td>5</td>
<td>0.991339</td>
<td>0.990847</td>
<td>0.991711</td>
<td>0.992389</td>
</tr>
<tr>
<td>7</td>
<td>0.995792</td>
<td>0.996384</td>
<td>0.99634</td>
<td>0.996889</td>
</tr>
</tbody>
</table>

4.3.2 Explanations

After examining the results of the bursty traffic simulations, many different conclusions were made when compared to the other two traffic types. One difference which was not expected was the improved performance of the Helical architecture. In the majority of different bursty traffic cases, the Helical switch performed better than both the Knockout and the Sunshine architectures, as mentioned in conclusions 2, 3, and 5. There are several reasons for this improved performance. The main reason for this increased performance is that the bursty traffic parameter sets simulated possessed low equivalent loads. For the purpose of comparison, the equivalent load of a particular bursty traffic scenario can be defined as

$$ Equivalent \ Load = \frac{Mean \ Burst \ Length}{Mean \ Burst \ Length + Mean \ Interarrival \ Time} \quad Eq. \ 48 $$
Effectively, this term attempts to describe the probability that a time slot will be filled by a burst cell arriving at a particular input port of the switch, much like the parameter $p$ used to describe random traffic cell arrival. The equivalent load term neglects the fact that the cells in question belong to bursts and assumes that each cell arrives independently of other cells at the input port. Table 19 displays the equivalent loads of the bursty traffic parameter sets simulated. As can be seen from the table, all the bursty traffic parameter sets simulated except for {5, 4} possessed an equivalent load lower the lowest input load $p$, {0.5}, used for random traffic.

As was discovered in the random traffic simulation analysis, the Helical switch performed poorly at higher input loads (equivalent loads) because of the concentrator's inability to process all the arriving real cells in a given time slot. Real cells which could not be processed were delayed, which caused the buffers they were stored in to overflow and discard cells just arriving at their inputs. Because the equivalent loads of the bursty traffic are very low, this situation occurs much infrequently than at the the higher random input loads, and more cells can be successfully routed through the switch.

<table>
<thead>
<tr>
<th>Interarrival Time</th>
<th>Mean Burst Length (cells)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>0.43</td>
</tr>
<tr>
<td>6</td>
<td>0.33</td>
</tr>
<tr>
<td>8</td>
<td>0.27</td>
</tr>
<tr>
<td>10</td>
<td>0.23</td>
</tr>
<tr>
<td>12</td>
<td>0.20</td>
</tr>
<tr>
<td>14</td>
<td>--</td>
</tr>
<tr>
<td>15</td>
<td>--</td>
</tr>
<tr>
<td>16</td>
<td>--</td>
</tr>
<tr>
<td>20</td>
<td>--</td>
</tr>
<tr>
<td>25</td>
<td>--</td>
</tr>
<tr>
<td>30</td>
<td>--</td>
</tr>
</tbody>
</table>

This does not entirely explain why the Helical switch performed better than the other two architectures, however. The remainder of the answer lies in the multipath nature of the Helical switch. Because the of its multipath nature, the Helical switch distributes cells across many different buffers as they move through the different stages of the switch. The buffers are not designated to one specific set of internal connections between an input and output port, but to many (except in the first stage). This is the helical buffer concept explained in section 2.3. A cell travelling through the an 8x8 Helical switch may be stored in a total of $B_1 + N/2 * B_2 + N/4 * B_3$ buffers. For the $H_1$ configuration with $B = \{3, 3, 6\}$, this is equal to 27 buffers and for the $H_2$ configuration with $B = \{6, 6, 12\}$, this is equal to 54 buffers. For cells submitted to the Knockout and Sunshine architectures, however, only 12 buffers were available in the buffer 1 configuration and 20 in the buffer 2 configuration for buffering the bursts destined to the same output port. Because of this partial buffer sharing within the Helical architecture and the low equivalent traffic loads at the input
ports, the Helical architecture demonstrated lower CLP than the Knockout and Sunshine architectures for increasing interarrival times, regardless of burst length.

It is also interesting to note that this conclusion was suggested by the results of the random traffic simulations. As was noted in conclusion 1 of that section, the rate at which the Helical CLP decreased was much steeper than for the other two architectures (see figures 46 and 47) in both buffer configurations. At some point around the 60% input load, the Helical CLP performance exceeded the performance of the other two architectures. Thus, this indicated the Helical switch might perform better than the other two architectures for input loads lower than 50% or 60%. This same crossover situation occurred under bursty traffic. Examining figure 68, the (5,4) bursty parameter set represents an equivalent load of 0.56. At this point the Helical H, CLP performance is slightly worse than the Knockout and Sunshine architectures. As the equivalent load decreases from this point, the Helical H, CLP performance quickly surpasses that of the other architectures. This general trend and conclusion seems consist will all the collected data.

Conclusion 5 also needs further investigation. For smaller length bursts, the Sunshine architecture with more parallel Banyan routing networks (S7 has 4) performed better than those configurations with fewer Banyan networks (S6 only possessed 3) as the interarrival time between bursts increased. This trend reversed, however, when burst length became longer. Configurations with fewer Banyan routing networks performed better than those with more. This situation also occurred in the buffer 1 configurations between S1 and S2. When the burst lengths were small, Sunshine configurations with larger numbers of parallel Banyan networks (S7) performed better because the overall architecture was capable of delivering more coincident bursts destined to the same output port to the corresponding output buffer simultaneously without requiring cells from those bursts to be recirculated. Because of the short length of the burst, the total number of cells in the coincident bursts was not great enough to exceed the capabilities of the destination output buffer. After the completion of the bursts, the output port would then process the stored burst cells in the output buffer before the next batch of bursts reached the switch. On the other hand, the Sunshine configurations with fewer Banyan networks (S6) performed worse than those with more Banyan networks for shorter length bursts. When the number of coincident bursts destined to the same output port exceeded the number of parallel Banyan routing networks (more likely in the S6 configuration than in the S7 configuration), some of the burst cells would be diverted to the recirculator. The recirculated burst cells would arrive at the Batcher unit in the next time slot only to compete with cells belonging to the same bursts and destined for the same output port. Thus,
the recirculator, within just a few time slots, would become inundated with recirculation requests with too few recirculating lines to process all the requests. Cells would then be discarded.

For larger burst lengths, however, this situation reversed. Configuration \( S_6 \) with only 3 Banyan networks performed better than configuration \( S_7 \) with 4 Banyan networks. This is because the configuration with more Banyan networks, \( S_7 \), was able to deliver the cells of up to four bursts destined to the same output port simultaneously. Because of the longer length of the bursts, though, the total number of cells contained within the group of bursts exceeded the size of the buffer at that output port. For example, if four bursts arrived together heading for output port one, there would be approximately 28 cells contained within those bursts for an average burst length of seven. Since only 20 cell buffers were present at each output port in the \( S_6 \) and \( S_7 \) configurations, some of those cells would have to be discarded due to buffer overflow. The \( S_6 \) configuration, however, was only capable of delivering cells from three of the four bursts to the same output port at the same time. The remaining burst cells would be recirculated. As indicated by the raw simulation data, more cells were lost at the recirculator in the \( S_6 \) configuration than in the \( S_7 \) configuration for this very reason. The \( S_7 \) configuration, however, lost significantly more cells at the output buffer than the \( S_6 \) configuration. By redirecting some of the burst cells away from the output buffer to the recirculator, fewer cells were lost at the output buffer in the \( S_6 \) configuration.

Lastly, the following table 20 summarizes the negative impact bursty traffic has on the sustainable, equivalent input load for a given CLP when compared to other traffic types like random traffic.

<table>
<thead>
<tr>
<th>( \log_{10} ) CLP</th>
<th>Knockout ( K_s )</th>
<th>Helical ( H_s )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Random</td>
<td>Bursty</td>
</tr>
<tr>
<td>-3</td>
<td>0.89</td>
<td>0.38</td>
</tr>
<tr>
<td>-4</td>
<td>0.82</td>
<td>0.26</td>
</tr>
</tbody>
</table>

### 4.4 Switch Complexity

Even though a proposed ATM architecture may have a long list of appealing features and generate favorable performance results under simulations, implementing the architecture in VLSI may introduce new problems. Primarily, these problems are complexity problems associated with the physical factors which describe the implementation of the ATM switching architecture in terms of the required number of transistor and I/O pads. It is thus important to examine the complexity of the three studied architectures to determine whether or not any of the architectures may or may not be practically implemented in VLSI technology and how the complexity of each architecture compares to the other architectures for certain simulated levels of performance. This section will
attempt to generate rough complexity estimate for each of the three architectures in order to
determine whether or not a particular architecture's performance may be negated by a high
complexity estimate. Also, the complexity of each architecture will be estimated for several
additional cases which were not simulated in this thesis to provide a glimpse of how the complexity
of three architectures changes for larger switch sizes. In each estimate, the input and output
modules will not be considered since the internal complexity of these units and their interaction with
the SMM and CAC modules would require them to be implemented in a separate VLSI package.

The first step in developing the complexity estimate in terms of the number of transistors is to define
some of the basic values and terms associated with the three ATM switching architectures studied.
Table 21 lists the major building blocks and variables needed to implement the Knockout, Sunshine,
and Helical switches. The ATM cell size is needed to determine the number of memory bits
required to store ATM cells in buffers. The switch size is also important for establishing the number
of switching elements required to implement the various subcomponents of the three architectures
like the Banyan routing networks in the Sunshine architecture and the concentrator unit in the
Knockout switch. Some of the basic building blocks needed to implement the core of the different
architectures are a one bit adder circuit, a one bit dynamic shift register, and a one bit static
register. The adder circuit is needed to implement the Running Adder Network (RAN) which is part
of the concentrator unit in the Helical switch. In addition, a one bit, static register circuit will be
needed to implement the storage of the counter value associated with each running adder. In
addition, since cells are serially transmitted through each architecture, single bit, dynamic shift
register circuits could be combined together to make a large shift register capable of storing
(buffering) a single ATM cell. Combining these large shift registers together forms the FIFO
structures needed to buffer arriving ATM cells destined for the same output port. In actual
implementation, the buffers would not be made exclusively from large shift register circuits, but for
the purpose here, the large shift register concept will suffice to create a estimate of the number of
transistors needed to build the required cell buffering. Finally, many of the networks and
components forming the three different ATM architectures are based upon the contention switch
described in section 1.4.5.2.5. Table 21 lists the complexity of these basic building blocks along
with the variable names with which they are referred to by.
Table 21 Definitions and Basic Building Blocks for Complexity Estimate

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Variable Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATM Cell Size</td>
<td>424 bits</td>
<td>CELL</td>
</tr>
<tr>
<td>Simulated Switch Size</td>
<td>8</td>
<td>N</td>
</tr>
<tr>
<td>Dynamic Shift Register Bit</td>
<td>10 transistors</td>
<td>DSRB</td>
</tr>
<tr>
<td>Static Register Bit [21]</td>
<td>15 transistors</td>
<td>SRB</td>
</tr>
<tr>
<td>One Bit Adder [21]</td>
<td>25 transistors</td>
<td>ADD</td>
</tr>
<tr>
<td>Contention Switch [22]</td>
<td>112 transistors</td>
<td>Y</td>
</tr>
<tr>
<td>XOR, DEMUX</td>
<td>10 transistors</td>
<td>XOR, DEMUX</td>
</tr>
</tbody>
</table>

From the basic building blocks and definitions listed in table 21, the number of transistors required to build a single ATM cell buffer was computed. In addition, the number of transistors needed to implement one bit of a running adder was also computed. The number of bits needed for each running adder belonging to a particular concentrator depends upon the number of input/output lines at the concentrator has, which in turn depends on the stage at which the concentrator unit is located in the Helical switch. It is assumed that the running adder bit can be constructed from a simple one bit static register and adder circuit, ignoring other support logic. These values are listed in table 22 along with variables names with which they will be referred to by.

Table 22 Larger Building Blocks for Architecture Complexity Estimates

<table>
<thead>
<tr>
<th>Building Block</th>
<th>Transistors</th>
<th>Variable Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>One ATM Cell Buffer</td>
<td>$CELL \times DSRB = 4240$</td>
<td>$X$</td>
</tr>
<tr>
<td>Running Adder per bit</td>
<td>$ADD + SRB = 40$</td>
<td>$Z$</td>
</tr>
</tbody>
</table>

4.4.1 Knockout Switch

The basic building block of the Knockout switch is the Bus Interface Unit (BIU) seen in figures 19 and 20 in sections 2.1.2 and 2.1.2.2 respectively. The BIU is composed of four major components:
cell filters, a concentrator, a shifter unit, and cell buffers. Each BIU has $N$ cell filters. The cell filter can be implemented with a single 2x2 contention switch (made of $Y$ transistors) using only one of its inputs and outputs. The concentrator unit as described in section 2.1.2.3 is composed of contention switches and single bit delay units. A worst case estimate of the number of contention switches needed is $N^*L$ [9]. Since this is a worst case estimate, the complexity of the single bit delay unit is not included. The shifter unit can be implemented from $L/2 \log_2(L)$ contention switches arranged in an omega (Banyan style) network. A simple state machine controls the shifter unit, but its complexity will be ignored here. Finally, each BIU has $B$ cell buffers at the output of the shifter unit. Table 23 lists the number of transistors required for each BIU component.

Since there are $N$ BIUs, the total complexity estimate in terms of the number of required transistors can be expressed as the sum of the individual complexity estimates forming the BIU times $N$.

$$
Knockout \ Switch \ Complexity = N \cdot \left[ (N \cdot Y) + (N \cdot L \cdot Y) + \left( \frac{L}{2} \cdot \log_2(L) \cdot Y \right) + (B \cdot X) \right] \quad \text{Eq. 49}
$$

<table>
<thead>
<tr>
<th>Subcomponent</th>
<th>Complexity (transistors)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell Filter</td>
<td>$N \cdot Y$</td>
</tr>
<tr>
<td>Concentrator</td>
<td>$N \cdot L \cdot Y$</td>
</tr>
<tr>
<td>Shifter</td>
<td>$\frac{L}{2} \cdot \log_2(L) \cdot Y$</td>
</tr>
<tr>
<td>Buffer</td>
<td>$B \cdot X$</td>
</tr>
</tbody>
</table>

Table 23 lists the complexity of the Knockout switch for the $K_1$ and $K_2$ configurations in addition to several other large switch cases. The estimate has been rounded to the nearest 50k transistors.
Table 24 Knockout Switch Complexity Estimate for Various Configurations in transistors

<table>
<thead>
<tr>
<th>Configuration</th>
<th>K₁</th>
<th>K₂</th>
<th>Case 1</th>
<th>Case 2</th>
<th>Case 3</th>
<th>Case 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Ports</td>
<td>8</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
<td>128</td>
</tr>
<tr>
<td>Concentrator Outputs</td>
<td>4</td>
<td>5</td>
<td>12</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Buffers</td>
<td>12</td>
<td>20</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>Cell Filters</td>
<td>896</td>
<td>896</td>
<td>1792</td>
<td>3584</td>
<td>7168</td>
<td>14336</td>
</tr>
<tr>
<td>Concentrator</td>
<td>3584</td>
<td>4480</td>
<td>21504</td>
<td>57344</td>
<td>114688</td>
<td>229376</td>
</tr>
<tr>
<td>Shifter</td>
<td>448</td>
<td>1344</td>
<td>3584</td>
<td>3584</td>
<td>3584</td>
<td>3584</td>
</tr>
<tr>
<td>BIU Complexity</td>
<td>4928</td>
<td>6720</td>
<td>26880</td>
<td>64512</td>
<td>125440</td>
<td>247296</td>
</tr>
<tr>
<td>Complexity w/o Buffers</td>
<td>39424</td>
<td>53760</td>
<td>430k</td>
<td>2046k</td>
<td>8028k</td>
<td>31653k</td>
</tr>
<tr>
<td>Buffer Complexity</td>
<td>407k</td>
<td>678k</td>
<td>2713k</td>
<td>5427k</td>
<td>10854k</td>
<td>21708k</td>
</tr>
<tr>
<td>Switch Complexity</td>
<td>450k</td>
<td>750k</td>
<td>3150k</td>
<td>7500k</td>
<td>18900k</td>
<td>53400k</td>
</tr>
</tbody>
</table>

4.4.2 Sunshine Switch

The Sunshine architecture is composed of 6 basic components. The first component is the Batcher network. This is a sorting network composed entirely of contention switches. The number of contention switches needed to construct this network was derived in Eq. 21 in section 2.2.2.3. One important note is that the number of input ports to the Batcher sorter network, N', is equal to the number of switch input ports N and the number of recirculating lines T, and it must be a power of 2. After the Batcher network comes the trap network, which is composed of single bit comparators, shift registers, and multiplexers. The comparators are used to compare adjacent output lines from the Batcher network to determine if more than k output lines have cells destined for the same switch output port. If so, the trap unit must swap the output destination and priority fields in the cell header. The shift register is used to delay the leading output address field in the header. The shift register must, therefore, have as many bits as there are in the address field. The multiplexer is used to reintegrate the cell with its control fields swapped. The trap network is composed of N' lines. The trap network outputs N' lines to the concentrator network, which is another Batcher sorting network with the same number of contention switches described by Eq. 21, each of which has Y gates. After the concentrator, cells are passed to the selector unit, which determines whether cells go to the recirculators or the Banyan networks. The input lines to this unit may be implemented with a simple demultiplexer. Finally, the recirculators lines must delay cells which are
being recirculated to arrive at the batcher network at the beginning of the next time slot. It must also swap the two fields interchanged previously by the trap network. Finally, it must decrement the priority field. Thus each recirculator line must have a shift register to delay the recirculated cell, a shifter register and multiplexer to interchange the two control fields, and an adder/decrementer circuit to change the value of the priority field. It is assumed for this discussion that a priority field has a length of five bits (32 priority levels), and thus the shift register responsible for swapping the control fields must be 5 bits long. The shifter register used to delay the recirculated cells is assumed be half a cell length as an average estimate. The last component, the Banyan routing networks, are each composed of \((N/2)\log_2 N\) contention switches. Table 25 summarizes the complexity of each unit in the Sunshine switch. Adding these estimates together plus the estimate for the required number of buffers is the total complexity estimate for the Sunshine architecture.

Table 26 lists the complexity of the Sunshine switch for configurations \(S_1\), \(S_2\), \(S_6\), and \(S_7\) in addition to several other large switch cases. All estimates have been rounded to the nearest 50k transistors. As can be seen when compared to the Knockout switch, the Sunshine architecture seems to have a reduced level of complexity when the architecture is examined without the buffer complexity included. This complexity difference grows dramatically as switch size is increased. For a switch size of 32x32, the Sunshine complexity is 232k transistor gates while the Knockout switch complexity is 2064k gates, nearly a 10 fold difference.
<table>
<thead>
<tr>
<th>Component</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batcher Network</td>
<td>$\frac{N}{4} \cdot (\log_2 N) \cdot (1 + \log_2 N) \cdot Y$</td>
</tr>
<tr>
<td>Trap Network</td>
<td>$N \cdot (\lceil \log_2 N \rceil \cdot X + \text{XOR} + \text{DEMUX})$</td>
</tr>
<tr>
<td>Concentrator Network</td>
<td>$\frac{N}{4} \cdot (\log_2 N) \cdot (1 + \log_2 N) \cdot Y$</td>
</tr>
<tr>
<td>Selector</td>
<td>$Y \cdot N$</td>
</tr>
<tr>
<td>Banyan Networks</td>
<td>$k \cdot (\frac{N}{2} \cdot \log_2 N) \cdot Y$</td>
</tr>
<tr>
<td>Buffers</td>
<td>$B \cdot X$</td>
</tr>
<tr>
<td>Recirculators</td>
<td>$\frac{T}{2} \cdot X$</td>
</tr>
</tbody>
</table>

Table 25 Complexity of Sunshine Switch Components
4.4.3 Helical Switch

Developing a complexity estimate for the Helical switch involved estimating the complexity of fewer components than other two architectures. The Helical switch contains only three main components which need to be examined: cell buffers, broadcast units, and concentrators. The complexity of the cell buffers can be quickly computed exactly as was done in the last two sections.

\[
\text{Buffer Complexity} = X \cdot \left( \sum_{k=1}^{n} 2 \cdot N \cdot B_k \right) \quad \text{Eq. 50}
\]

where \(B_k\) represents the depth of the buffers at stage \(k\) of the switch and \(n = \log_2 N\). The broadcast units which supplies the buffers with cells can be implemented with a modified contention switch. Each of the \(k\) stages of switch contains exactly \(N\) broadcast units, each made of a contention switch with a complexity of \(Y\).

\[
\text{Broadcast Unit Complexity} = N \cdot (\log_2 N) \cdot Y \quad \text{Eq. 51}
\]
The concentrator unit was the most difficult unit to calculate complexity for because its size and number of input and output lines varies at each stage of the switch. To determine the complexity of the concentrator, its two subcomponents, the Running Adder Network and the reverse Banyan network, were examined separately. The complexity of the RAN is described by

\[
RAN \text{ Complexity} = \sum_{k=1}^{n} \left( 2^k \cdot (2^{n-k+1}) \cdot (n - k) \cdot Z \right) \text{ where } n = \log_2 N \quad \text{Eq. 52}
\]

The first term in the summation determines the number of concentrators at stage \( k \). The second term specifies the number of running adders (the number of inputs) within each concentrator at stage \( k \). The last significant term indicates the number of bits which must be encoded into a routing tag to be used by the reverse Banyan network following the RANs at that stage, and \( Z \) is the estimate of the number of gates needed for each of those bits to construct the running adder. The other component is the reverse Banyan network. The basic Banyan network is composed of \((N/2) \log_2 N\) switching elements where \( N \) is the number of input/output ports. The following equation sums the number of contention switches required by all the concentrators at a stage \( k \) for all the stages in the switch.

\[
Reverse \text{ Banyan Complexity} = \sum_{k=1}^{n} \left[ 2^k \cdot \left( \frac{2^{n-k+1}}{2} \log_2 (2^{n-k+1}) \right) \right] \cdot Y \text{ where } n = \log_2 N \quad \text{Eq. 53}
\]

This equation can be simplified to

\[
Reverse \text{ Banyan Complexity} = \left( \sum_{k=1}^{n} 2^k \cdot (n-k+1) \right) \cdot Y \quad \text{Eq. 54}
\]

Summing all the complexity estimates for the three components yields the overall complexity estimate for the Helical switch. Table 27 lists the complexity of the Helical switch for the \( H_1 \) and \( H_2 \) configurations in addition to several other large switch cases. The estimate has been rounded to the nearest 50k transistors.
Table 27  Helical Switch Complexity Estimate for Various Configurations in
transistors

<table>
<thead>
<tr>
<th>Configuration</th>
<th>$H_1$</th>
<th>$H_2$</th>
<th>Case 1</th>
<th>Case 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Ports, N</td>
<td>8</td>
<td>8</td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td>Number of Stages, $n$</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Stage 1 Buffer Depth, $B_1$</td>
<td>3</td>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Stage 2 Buffer Depth, $B_2$</td>
<td>3</td>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Stage 3 Buffer Depth, $B_3$</td>
<td>6</td>
<td>12</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>Stage 4 Buffer Depth, $B_4$</td>
<td></td>
<td></td>
<td>20</td>
<td>12</td>
</tr>
<tr>
<td>Stage 5 Buffer Depth, $B_5$</td>
<td></td>
<td></td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>Broadcast Units</td>
<td>2688</td>
<td>2688</td>
<td>7168</td>
<td>17920</td>
</tr>
<tr>
<td>Concentrators</td>
<td>5376</td>
<td>5376</td>
<td>17920</td>
<td>53760</td>
</tr>
<tr>
<td>RANS</td>
<td>4480</td>
<td>4480</td>
<td>19200</td>
<td>81920</td>
</tr>
<tr>
<td>Complexity w/o Buffers</td>
<td>12544</td>
<td>12544</td>
<td>44288</td>
<td>154k</td>
</tr>
<tr>
<td>Buffer Complexity</td>
<td>815k</td>
<td>1628k</td>
<td>5427k</td>
<td>13025k</td>
</tr>
<tr>
<td>Total Switch Complexity</td>
<td>850k</td>
<td>1650k</td>
<td>5500k</td>
<td>13200k</td>
</tr>
</tbody>
</table>

4.4.4 Conclusions

The following table compares the approximate complexity estimates for the three architectures for the simulated configurations. The comparison is broken down by buffer 1 and 2 configurations.

Table 28  Comparison of the Complexity for the 3 Architectures

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Complexity w/o Cell Buffers</th>
<th>Complexity with Cell Buffers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knockout, $K_1$</td>
<td>39k</td>
<td>450k</td>
</tr>
<tr>
<td>Sunshine, $S_1$ &amp; $S_2$</td>
<td>27k</td>
<td>450k</td>
</tr>
<tr>
<td>Helical, $H_1$</td>
<td>13k</td>
<td>850k</td>
</tr>
<tr>
<td>Knockout, $K_2$</td>
<td>54k</td>
<td>750k</td>
</tr>
<tr>
<td>Sunshine, $S_6$ &amp; $S_7$</td>
<td>29k</td>
<td>750k</td>
</tr>
<tr>
<td>Helical, $H_2$</td>
<td>13k</td>
<td>1650k</td>
</tr>
</tbody>
</table>
The following list represents the major conclusions which can be drawn about the complexity of the three architectures from the complexity estimates generated.

1. The cell buffers constituted the majority of the switch complexity estimates.

2. The Helical switch used twice as many cells buffers as the Knockout and Sunshine architectures for the buffer 1 and 2 switch configurations; the Helical switch complexity estimates were approximately twice as large as those for the Knockout and Sunshine switches for only moderately similar simulation performance.

3. Not considering cell buffers, the Helical switch had the lowest complexity estimate compared to the Knockout and Sunshine switches.

4. Not considering cell buffers, the Sunshine switch complexity was approximately 30% lower than the Knockout switch complexity for the buffer 1 configurations and 50% lower for buffer 2 configurations for nearly identical simulation performance. Also, the Sunshine switch complexity increased at a much slower rate than did the Knockout switch for increasing N.

5. For the Sunshine Network, the number of trap lines can be increased with negligible increase in complexity unless the additional recirculator lines causes the needed number of Batcher input/output lines to exceed a number which is a power of 2 (i.e. a Sunshine configuration with 32 inputs and 1 recirculator would require 64 input/output lines as would a configuration with 32 inputs and 32 recirculator lines).

Several other conclusions and comments should be mentioned about complexity of the three different architectures. First, it is quite clear that the buffering requirements for larger switches is the primary factor in the complexity estimates. Because of this fact, the Helical switch has two major disadvantages. First, the internal distribution of cells at each stage of the architecture forces the Helical buffering to be built into the Helical switch IC. If instead external buffering is to be used, the architecture must be divided into many ICs which group together components at a single stage of the architecture. Thus, one IC may be used to implement the concentrator and broadcast units at the first stage of the switch. By arranging the multiple ICs around external memory, the full Helical architecture can be created. This task, however, is difficult and impractical. Within the Helical architecture, the concentrator units have a varying numbers of input/output lines depending on what stage of the architecture they are located. Thus, multiple IC designs would be required to account for all the different sized concentrator units needed to implement the full Helical switch architecture. Because of these problems, this architecture cannot be easily implemented in VLSI and then laid out on PCB. The Knockout and Sunshine architectures, on the other hand, can easily offload cell buffering to external, low cost memory devices like DRAM with the help of some external control circuitry because of their use of the output buffering technique. The actual switching architecture can then be constructed in a single IC package without the cells buffer, allowing larger
sized switches to be built. Second, the Helical switch requires more buffering than the Sunshine and Knockout architectures to achieve the same levels of performance as indicated by the simulation results. In some cases, no amount of buffering would improve the performance of the Helical switch (like above an 83% random input load). In the buffer 1 architecture configurations, the Sunshine architectures were configured with a total of $12 \times 8 = 96$ cell buffers. The Helical switch configuration, however, contained twice that number of cell buffers, 192. Combining the need for large buffers with the first problem makes the Helical switch appear to be a less attractive solution to the ATM switching problem.

4.5 Other Characteristics

There are three other characteristics related to the switching architectures which are important to mentioned for each of the architectures studied in this thesis. These characteristics are broadcast and multicast capabilities, modularity, and fault tolerance. The following sections will briefly describe these characteristics as they are related to the Knockout, Sunshine, and Helical architectures.

4.5.1 Broadcast/Multicast

Many future broadband services like video on demand will require transporting data from a single service provider to many end users. Thus, each switching architecture must provide broadcast and multicast features to support these types of services. Only one of the three architectures studied readily supports these functions, the Knockout switch. Internally, each input port of the Knockout switch broadcasts a copy of an arriving cell to the bus interface unit (BIU) of each output port. Normally, the cell filters inside the bus interface unit block the progress of cells not destined for the output port to which the bus interface unit is connected. With the introduction of more complex cell filters, broadcast and multicast functions can be easily supported. The new cell filters would have to be dynamically programmable to recognize special broadcast/multicast routing identifiers established by the call admission control module. Some new delay would be added to the routing of cells of these new filters because extra time would be required at the complex cell filters to check whether or not the cell belonged to a broadcast/multicast connection. The Sunshine and Helical switches, because of there basic space division architecture however, are not inherently suited to support broadcast/multicast functions. To achieve this functionality, special copy networks would have to proceed the main Sunshine and Helical switching architectures to copy the cells marked to broadcast/multicast groups to the appropriate input ports of the switching architectures. Many
complexities are associated with these copy networks. For instance, additional buffering must be provided to compensate for the cases when cells belonging to point-to-point ATM connections arrive concurrently with a cell being broadcast to all switch output ports. The tradeoff between the addition of a copy network to the Sunshine and Helical switch architectures and the increased complexity of the cell filters in the Knockout switch is not clear. The complexity of the cell filters grows as a factor \( N^2 \) whereas copy networks, much like the Batcher sorting network described in section 2.2.2.3, have complexities on the order of \( N(\log_2 N) \). If the complexity of the new cell filters for the Knockout switch is too high, using copy networks with the Sunshine and Helical architectures may be more practical, not considering other factors like performance.

4.5.2 Modularity

Another factor important to choosing a switching fabric architecture is whether or not the architecture is easily expandable in size, or is modular, from a given switch size \( N \). The Knockout switch can be expanded linearly from a \( N \times N \) to \( AN \times AN \), where \( A \) is an integer number [9]. Each concentrator unit in this configuration has its number of inputs increased from \( N \) to \( N+L \), making a \( N+L \) to \( L \) concentrator. Then, \( A \) rows of \( AN \) concentrators and cell filters are used to route data to only \( AN \) shifter units and buffers. An example of this layout is shown in figure 82 for a \( 2N \times 2N \) expansion. The Sunshine and Helical architectures, however, cannot be expanded as easily from a basic switching block of size \( N \) to make large switch sizes as can be in the Knockout switch. One method of expanding these networks would be to increase the basic switch size \( N \). The size to which \( N \) can grow, however, is limited by VLSI design constraints. Another method of expanding the switch size would be to create a new MIN (like a Banyan network) in which the basic building block would be the \( N \times N \) Sunshine or Helical switch. This method could also be applied to the Knockout switch.

4.5.3 Fault Tolerance

![Figure 82 Knockout Switch 2Nx2N expansion](image-url)
In ATM architectures, there is a need for fault tolerant designs. Failure within a switching architecture can cause one or hundreds of connections to be disrupted. Several common techniques are used to account for system faults. One expensive technique is the creation of a duplicate, standby switching fabric which takes over the functions of the primary switching fabric in the event of a fault. Other solutions include using redundant subcomponents within the switching architecture. Of the three architectures studied, the Knockout switch is the most fault tolerant since it is designed around a basic functional component, the bus interface unit. Any fault in a BIU only causes the disruption of traffic at either its inputs or outputs, but it does not affect the remainder of the BIUs in the switch. Because of the design of the Knockout switch, a reserve BIU may be connected to the N broadcast buses within the switch to take over the functions of a failed BIU. The Sunshine and Helical switches are not as fault tolerant as the Knockout switch, however, due to their space division architecture. In the Sunshine switch, the failure of a contention switch within any of the internal networks like the Batcher, concentrator, or Banyan routing networks can affect numerous input/output connections. Furthermore, these faults may be difficult to isolate and even more difficult to correct, depending on where the fault occurs. If the fault occurs in one of the k parallel Banyan routing networks, then the afflicted network could be simply disabled at the cost of reduced performance or replaced with a reserve Banyan network. If the fault occurs within the Batcher or concentrator networks at the contention switch level, the number of affected connections depends on which stage in the network the failure occurred. Backup contention switches cannot be incorporated into these networks since their interconnections would depend on the position of the failed contention switch. Thus, the most practical fault tolerant technique which could be used for the Sunshine architecture would have be the use of redundant Sunshine switching fabrics. Considering the lower complexity of the basic Sunshine architecture without buffers when compared to the Knockout switch architecture, this is a viable option for much larger switch sizes. This scenario is also true of the Helical architecture. A fault in either the broadcast units or the concentrators within the Helical switch will interrupt a varying number input/output connections depending on which stage of the switch the failure occurs. Again, the use redundant subcomponents within the Helical switch is not feasible because the interconnections and characteristics of those components also depends on the stage where the failure occurred.
5 Conclusions

5.1 ATM Architectures

After completing the simulation of the Knockout, Sunshine, and Helical ATM switches under various architectural configurations and traffic conditions, many general conclusions about the architectures can be drawn from the simulation results. These conclusions account for the significant issues related to ATM switching architectures, like CLP performance, cell delay, complexity, etc. Before making these conclusions, however, they must first be qualified. Overall, a reasonable baseline comparison was performed between the three architectures studied. Care was taken to make sure results were statistically meaningful, and many differences seen (and not seen) between the different switch architectures were confirmed by both categories of architecture configurations simulated, buffer one and two configurations. It is always possible that one switching architecture could be designed to perform better than another under the same simulation conditions, but to eliminate this possibility, the complexity of each architecture, in terms of the number of transistors required to implement it, was computed and compared to that of the other architectures. The comparison between the different Sunshine configurations, however, was extremely limited due to the small number of configurations simulated and the small variation in the architecture parameters between those configurations. More Sunshine configurations and larger parameter variations between those configurations would have been desirable. Furthermore, it should be noted that the results obtained are sensitive to both architecture configurations and the input traffic type. Several steps were taken to reduce the sensitivity of the results to the architecture configurations. First, the configurations were simulated under two broad buffer configurations to verify trends seen in the results. Second, the configurations were chosen so that they would have relatively the same CLP performance under one specific traffic pattern. When differences in the results were seen between the configurations under different traffic patterns, these differences or lack of were attributed to the architectures themselves and not to the architectural configurations selected for the three switches. In regards to the sensitivity of the results to the input traffic pattern, the results obtained were not compared across different traffic types, but were compared only within a particular traffic type. The traffic types simulated were chosen to represent a broad group of data traffic which would be experienced in a real ATM network. The general trends between the architectures are believed to be accurate, but in the event that a new traffic type emerges, it must be explored and simulated for the three architectures to verify the trends witnessed in this thesis are still true in general.
Upon review of the simulation results collected, out of the three switching architectures, the Sunshine architecture has the best performance for a given switch size. When the Sunshine architecture is designed with three, but preferably four, Banyan routing networks, its performance was equivalent to, or better than the Knockout and Helical architectures for random and constant bit rate traffic. Increasing the number of Banyan networks above four, however, achieves little performance gain because only an extremely small percentage of submitted cells take advantage of the small recirculator system within the architecture. This fact has more significance under bursty traffic, where the recirculator system plays a more significant role diverting a portion of the arriving cell bursts away from output buffers until the bursts subside and the cell buffers can process the remaining burst cells temporarily stored in the recirculator. The Sunshine switch did not, however, outperform the Helical switch when subjected to bursty traffic. After realizing that the Helical switch configurations simulated were equipped with twice the amount of cell buffering though, it becomes very unlikely that the Helical switch could outperform the Sunshine architecture equipped with an equivalent number of cell buffers under the same bursty traffic. The Sunshine architecture, however, has several disadvantages which must be considered. First, it is not a fault tolerant design. A fault within the Sunshine architecture can affect a variable number of connections. Second, it does not readily support broadcast and multicast functions which will be required by future ATM networks. This problem can be overcome by the addition of a copy network preceding the Sunshine architecture. Such a solution, however, increases switch complexity.

If switch performance and complexity are secondary concerns to modularity, fault tolerance, and support of broadcast/multicast functions, then the Knockout switch would be the architecture of choice. The Knockout switch demonstrated performance levels very comparable to the Sunshine architecture. Like the Sunshine switch, the Knockout switch's CLP performance was worse when compared to the Helical switch under bursty traffic, but for the same reason mentioned earlier. Unlike the other two architectures, a fault within the Knockout architecture only affects a single input or single output port. Furthermore, by redesigning some of the internal components, the Knockout switch is naturally suited to support broadcast and multicast functions. Lastly, the design of the Knockout switch would allow for easy growth based upon a smaller Knockout switch size in addition to an easy VLSI packaging and printed circuit board (PCB) layout.

The Helical switch would not be recommended for physical implementation for many reasons. First, the architecture is limited to a throughput of only 83% under high random and CBR traffic loads. To obtain a similar CLP from the Helical architecture as was demonstrated by the Knockout and Sunshine architectures, it would have to be operated a lower, maximum input load. This situation
is not desirable due to the high cost of implementing the Helical switch because of the larger quantity of buffering needed to reach these equivalent levels of performance. Second, the cell delay times of this architecture were longer than those of the Sunshine and Knockout architectures. Third, this architecture, because of amount of cell buffers needed and their internal distribution, does not easily support VLSI implementation and PCB layout. Lastly, the Helical architecture does not support broadcast/multicast functions and is not particularly fault tolerant. For these reasons, using the Helical architecture for an ATM switching solution would not be a practical choice.

5.2 Problems

During the course of this thesis, many problems were encountered. One significant problem initially experienced was related to the stability of the ProModel simulator. For reasons which were not discovered, ProModel version 2 would periodically crash during the execution of simulations on the computers on which it was installed. When the newest release was obtained, ProModel version 3 pre-release, the frequency of this problem was dramatically reduced.

Another difficult problem faced during the thesis was the implementation of the Helical architectural model in ProModel. The ProModel environment did not provide convenient constructs and tools for implementing the logic required to mimic the behavior of the Helical concentrator unit as described in section 2.3.2.4. Specifically, the locations which performed the functions of the cell buffers could not be queried for information about what type of cell, real or dummy, was contained in the HOL position of that particular FIFO buffer. This information was, however, crucial to the operation of the concentrator unit. To overcome this problem, an elaborate signaling system was implemented in an array structure along with the use of specialized counters contained within each concentrator unit of the model. The HOL cells in the buffers servicing a particular concentrator would attempt to move into that concentrator during each time slot, at which point their type (real or dummy) could be determined. The completion of this move could be halted, however, depending on certain conditions established in the array structure set by the concentrator units. In addition, the development of this model was inhibited by the fact that the execution of logic at different locations within model did not follow a sequence which could be directly expressed by the model developer. Since the execution of logic for the Helical model was required to proceed in a specific order for correct operation, special logic was included in the model to force the simulator to execute the model logic in this required sequence. Consequently, the addition of the extra logic needed to correctly model the behavior of the Helical architecture had a negative impact on the model's
simulation speed, which was approximately 5 times longer than that of the Knockout and Sunshine models.

Other problems encountered include the amount of simulation time that was needed to collect the required data. When the three models were executed for the buffer 2 configurations, simulation times had to be increased for each simulation replication in order to measure enough occurrences of cells being discarded to obtain reliable results. Longer simulations times coupled with the number of architecture configurations and different traffic types and loads simulated placed large demands on computing time. In the end, 2500 to 3000 hours of computer simulation were used to gather the needed performance data. Lastly, significant time was invested converting the data output from the ProModel simulator to a format which could be used in Microsoft Excel. From Excel, graphical and tabular results were generated for comparing the performance of the three architectures.

5.3 Suggestions for Improvement

If this study had to be repeated, there are several areas which could use improvement or be changed completely. First, a different simulator should be chosen. Although ProModel sufficed for this thesis, it is not convenient and efficient enough for a long term study. One option would be to develop custom C models for each architecture in the study so that simulator overhead related to Windows could be eliminated and simulation execution times could be optimized. This would allow for the possibility of measuring lower CLPs than could be measured using the ProModel simulator. Second, it would be interesting to compare the performance of the Knockout and Sunshine architectures when configured with the same number of cell buffers that were used in the Helical architecture. Next, more configurations of the Sunshine architecture would be simulated, especially configurations utilizing larger numbers of recirculating lines since these lines tended to have negligible impact on the complexity of the Sunshine architecture but did improve the performance of the architecture to some measurable degree. Lastly, the traffic types simulated in this thesis cover only an extremely small subset of all the possible traffic types which will be experienced in B-ISDN. To ensure the proper performance of the studied architectures in the future B-ISDN, more traffic types should be simulated under various loading conditions.

5.4 Future Study
Reviewing the problems and work completed during this thesis, several ideas and areas for future research and investigation seem apparent. A recurring problem in this thesis was the amount of simulation time needed to measure CLP, or more specifically, the event of a cell being discarded. As CLP decreased towards more real world values, simulation times became unrealistic, at least with the chosen simulator. Research needs to be devoted to development of statistical techniques for measuring low probability events (like cell loss around $10^{-10}$) without requiring enormous simulation times (in this case, the time to submit and process $10^{11}$ to $10^{12}$ cells). If such techniques were developed, the relative performance trends of the three studied architectures could be confirmed for configurations which would achieve more realistic ATM performance requirements. Another area which deserves more attention is the development and simulation of the imbalanced traffic type on the three studied architectures, or any other ATM switching architecture. This traffic type is the most realistic traffic scenario considered since it models the possibility that many different traffic types will be supported on the same ATM switch. The main obstacle to imbalanced traffic research is the appropriate selection of the traffic parameters associated with this traffic type. Besides the exploring areas which expand beyond the original scope of this thesis, several areas within thesis which were not fully explored deserve possible attention. The Helical switch demonstrated some interesting performance results under bursty traffic, but that performance was not reflected in high load traffic cases for random and CBR traffic due to the nonblocking nature of the concentrator unit. It would be interesting to investigate the performance of this architecture if methods could be devised to remove the usage of dummy cells (which incur a high overhead) while preserving the multipath nature of the switch and to develop a more intelligent concentrator unit which would not necessarily be nonblocking. Lastly, a more thorough investigation is needed into the performance effects of varying the number of parallel Banyan networks and recirculating lines in the Sunshine architecture under various traffic types, particularly the effects of using a large number of recirculator lines.
References


Appendix A  Knockout Switch Bursty Model Text
Model Notes:

*Knockout Switch Model (Bursty) B3

A Simulation version of the Knockout ATM switching architecture.

*Buffer configuration: 1
*Buffer arrangement: 20 buffers per output port
*Concentrator L: 5
*Trafic pattern input: Bursty

*ccc = concentrator cell count
*bln = buffer location number

<table>
<thead>
<tr>
<th>Name</th>
<th>Cap</th>
<th>Units</th>
<th>Stats</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Ports 0 to 7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>input_port_0</td>
<td>1</td>
<td>1</td>
<td>None</td>
<td>Oldest,</td>
</tr>
<tr>
<td>input_port_1</td>
<td>1</td>
<td>1</td>
<td>None</td>
<td>Oldest,</td>
</tr>
<tr>
<td>input_port_2</td>
<td>1</td>
<td>1</td>
<td>None</td>
<td>Oldest,</td>
</tr>
<tr>
<td>input_port_3</td>
<td>1</td>
<td>1</td>
<td>None</td>
<td>Oldest,</td>
</tr>
<tr>
<td>input_port_4</td>
<td>1</td>
<td>1</td>
<td>None</td>
<td>Oldest,</td>
</tr>
<tr>
<td>input_port_5</td>
<td>1</td>
<td>1</td>
<td>None</td>
<td>Oldest,</td>
</tr>
<tr>
<td>input_port_6</td>
<td>1</td>
<td>1</td>
<td>None</td>
<td>Oldest,</td>
</tr>
<tr>
<td>input_port_7</td>
<td>1</td>
<td>1</td>
<td>None</td>
<td>Oldest,</td>
</tr>
</tbody>
</table>

Packet Filters 0 to 7

<table>
<thead>
<tr>
<th>Name</th>
<th>Cap</th>
<th>Units</th>
<th>Stats</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>pf_0</td>
<td>1</td>
<td>1</td>
<td>None</td>
<td>Oldest,</td>
</tr>
<tr>
<td>pf_1</td>
<td>1</td>
<td>1</td>
<td>None</td>
<td>Oldest,</td>
</tr>
<tr>
<td>pf_2</td>
<td>1</td>
<td>1</td>
<td>None</td>
<td>Oldest,</td>
</tr>
<tr>
<td>pf_3</td>
<td>1</td>
<td>1</td>
<td>None</td>
<td>Oldest,</td>
</tr>
<tr>
<td>pf_4</td>
<td>1</td>
<td>1</td>
<td>None</td>
<td>Oldest,</td>
</tr>
<tr>
<td>pf_5</td>
<td>1</td>
<td>1</td>
<td>None</td>
<td>Oldest,</td>
</tr>
<tr>
<td>pf_6</td>
<td>1</td>
<td>1</td>
<td>None</td>
<td>Oldest,</td>
</tr>
<tr>
<td>pf_7</td>
<td>1</td>
<td>1</td>
<td>None</td>
<td>Oldest,</td>
</tr>
</tbody>
</table>

Concentrators 0 to 7

<table>
<thead>
<tr>
<th>Name</th>
<th>Cap</th>
<th>Units</th>
<th>Stats</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>concentrator_0</td>
<td>1</td>
<td></td>
<td>None</td>
<td>Rand,</td>
</tr>
<tr>
<td>concentrator_1 8</td>
<td>1</td>
<td>None</td>
<td>Rand,</td>
<td></td>
</tr>
<tr>
<td>concentrator_2 8</td>
<td>1</td>
<td>None</td>
<td>Rand,</td>
<td></td>
</tr>
<tr>
<td>concentrator_3 8</td>
<td>1</td>
<td>None</td>
<td>Rand,</td>
<td></td>
</tr>
<tr>
<td>concentrator_4 8</td>
<td>1</td>
<td>None</td>
<td>Rand,</td>
<td></td>
</tr>
<tr>
<td>concentrator_5 8</td>
<td>1</td>
<td>None</td>
<td>Rand,</td>
<td></td>
</tr>
<tr>
<td>concentrator_6 8</td>
<td>1</td>
<td>None</td>
<td>Rand,</td>
<td></td>
</tr>
<tr>
<td>concentrator_7 8</td>
<td>1</td>
<td>None</td>
<td>Rand,</td>
<td></td>
</tr>
</tbody>
</table>

# Inactive Cell Trap
ict  INFINITE  1  Basic  Oldest, |

# Concentrator Reject Cell Trap
crc  24  1  Basic  Oldest, |

# Buffer Overflow Cell Trap
boct  INFINITE  1  Basic  Oldest, |

| buffer_0 | 100 | 5 | Basic | Oldest, FIFO, By turn |
| buffer_0.1 | 100 | 1 | None | Oldest, FIFO, |
| buffer_0.2 | 100 | 1 | None | Oldest, FIFO, |
| buffer_0.3 | 100 | 1 | None | Oldest, FIFO, |
| buffer_0.4 | 100 | 1 | None | Oldest, FIFO, |
| buffer_0.5 | 100 | 1 | None | Oldest, FIFO, |
| buffer_1 | 100 | 5 | Basic | Oldest, FIFO, By turn |
| buffer_1.1 | 100 | 1 | None | Oldest, FIFO, |
| buffer_1.2 | 100 | 1 | None | Oldest, FIFO, |
| buffer_1.3 | 100 | 1 | None | Oldest, FIFO, |
| buffer_1.4 | 100 | 1 | None | Oldest, FIFO, |
| buffer_1.5 | 100 | 1 | None | Oldest, FIFO, |
| buffer_2 | 100 | 5 | Basic | Oldest, FIFO, By turn |
| buffer_2.1 | 100 | 1 | None | Oldest, FIFO, |
| buffer_2.2 | 100 | 1 | None | Oldest, FIFO, |
| buffer_2.3 | 100 | 1 | None | Oldest, FIFO, |
| buffer_2.4 | 100 | 1 | None | Oldest, FIFO, |
| buffer_2.5 | 100 | 1 | None | Oldest, FIFO, |
| buffer_3 | 100 | 5 | Basic | Oldest, FIFO, By turn |
| buffer_3.1 | 100 | 1 | None | Oldest, FIFO, |
| buffer_3.2 | 100 | 1 | None | Oldest, FIFO, |
| buffer_3.3 | 100 | 1 | None | Oldest, FIFO, |
| buffer_3.4 | 100 | 1 | None | Oldest, FIFO, |
| buffer_3.5 | 100 | 1 | None | Oldest, FIFO, |
| buffer_4 | 100 | 5 | Basic | Oldest, FIFO, By turn |
| buffer_4.1 | 100 | 1 | None | Oldest, FIFO, |
| buffer_4.2 | 100 | 1 | None | Oldest, FIFO, |
| buffer_4.3 | 100 | 1 | None | Oldest, FIFO, |
| buffer_4.4 | 100 | 1 | None | Oldest, FIFO, |
| buffer_4.5 | 100 | 1 | None | Oldest, FIFO, |
| buffer_5 | 100 | 5 | Basic | Oldest, FIFO, By turn |
| buffer_5.1 | 100 | 1 | None | Oldest, FIFO, |
| buffer_5.2 | 100 | 1 | None | Oldest, FIFO, |
| buffer_5.3 | 100 | 1 | None | Oldest, FIFO, |
| buffer_5.4 | 100 | 1 | None | Oldest, FIFO, |
| buffer_5.5 | 100 | 1 | None | Oldest, FIFO, |
| buffer_6 | 100 | 5 | Basic | Oldest, FIFO, By turn |
| buffer_6.1 | 100 | 1 | None | Oldest, FIFO, |
| buffer_6.2 | 100 | 1 | None | Oldest, FIFO, |
| buffer_6.3 | 100 | 1 | None | Oldest, FIFO, |
buffer_6.4 100 1 None Oldest, FIFO,
buffer_6.5 100 1 None Oldest, FIFO,
buffer_7 100 5 Basic Oldest, FIFO, By turn
buffer_7.1 100 1 None Oldest, FIFO,
buffer_7.2 100 1 None Oldest, FIFO,
buffer_7.3 100 1 None Oldest, FIFO,
buffer_7.4 100 1 None Oldest, FIFO,
buffer_7.5 100 1 None Oldest, FIFO,
output_port_0 1 1 Basic Oldest, ,
output_port_1 1 1 Basic Oldest, ,
output_port_2 1 1 Basic Oldest, ,
output_port_3 1 1 Basic Oldest, ,
output_port_4 1 1 Basic Oldest, ,
output_port_5 1 1 Basic Oldest, ,
output_port_6 1 1 Basic Oldest, ,
output_port_7 1 1 Basic Oldest, ,

#Clock Control
clk_cntrl 1 1 Basic Oldest, ,

****************************************************************************************************

* Entities
****************************************************************************************************

Name   Speed (fpm) Stats
---     --------- --------

#This represents an ATM cell of 53 bytes of information, 5 bytes for the header and 48 bytes for data

cell 0 None
clock_tick 0 None

****************************************************************************************************

* Processing
****************************************************************************************************

<table>
<thead>
<tr>
<th>Entity</th>
<th>Location</th>
<th>Operation</th>
<th>Blk</th>
<th>Output</th>
<th>Destination</th>
<th>Rule</th>
<th>Move Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell</td>
<td>input_port_0</td>
<td>input_logic</td>
<td>1</td>
<td>cell</td>
<td>pf_0</td>
<td>FIRST 1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>input_port_1</td>
<td>input_logic</td>
<td>1</td>
<td>cell</td>
<td>pf_1</td>
<td>FIRST 1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>input_port_2</td>
<td>input_logic</td>
<td>1</td>
<td>cell</td>
<td>pf_2</td>
<td>FIRST 1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>input_port_3</td>
<td>input_logic</td>
<td>1</td>
<td>cell</td>
<td>pf_3</td>
<td>FIRST 1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>input_port_4</td>
<td>input_logic</td>
<td>1</td>
<td>cell</td>
<td>pf_4</td>
<td>FIRST 1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>input_port_5</td>
<td>input_logic</td>
<td>1</td>
<td>cell</td>
<td>pf_5</td>
<td>FIRST 1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>input_port_6</td>
<td>input_logic</td>
<td>1</td>
<td>cell</td>
<td>pf_6</td>
<td>FIRST 1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>input_port_7</td>
<td>input_logic</td>
<td>1</td>
<td>cell</td>
<td>pf_7</td>
<td>FIRST 1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>pf_0</td>
<td>pf_logic</td>
<td>1</td>
<td>cell</td>
<td>concentrator_0</td>
<td>FIRST 1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>pf_1</td>
<td>pf_logic</td>
<td>1</td>
<td>cell</td>
<td>concentrator_1</td>
<td>FIRST 1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>pf_2</td>
<td>pf_logic</td>
<td>1</td>
<td>cell</td>
<td>concentrator_2</td>
<td>FIRST 1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>pf_3</td>
<td>pf_logic</td>
<td>1</td>
<td>cell</td>
<td>concentrator_3</td>
<td>FIRST 1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>pf_4</td>
<td>pf_logic</td>
<td>1</td>
<td>cell</td>
<td>concentrator_4</td>
<td>FIRST 1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>pf_5</td>
<td>pf_logic</td>
<td>1</td>
<td>cell</td>
<td>concentrator_5</td>
<td>FIRST 1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>pf_6</td>
<td>pf_logic</td>
<td>1</td>
<td>cell</td>
<td>concentrator_6</td>
<td>FIRST 1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>pf_7</td>
<td>pf_logic</td>
<td>1</td>
<td>cell</td>
<td>concentrator_7</td>
<td>FIRST 1</td>
<td></td>
</tr>
</tbody>
</table>
cell   pf_1   pf_logic
1   cell   concentrator_0  FIRST 1
2   cell   concentrator_1  FIRST 1
3   cell   concentrator_2  FIRST 1
4   cell   concentrator_3  FIRST 1
5   cell   concentrator_4  FIRST 1
6   cell   concentrator_5  FIRST 1
7   cell   concentrator_6  FIRST 1
8   cell   concentrator_7  FIRST 1
9   cell   ict  FIRST 1

cell   pf_2   pf_logic
1   cell   concentrator_0  FIRST 1
2   cell   concentrator_1  FIRST 1
3   cell   concentrator_2  FIRST 1
4   cell   concentrator_3  FIRST 1
5   cell   concentrator_4  FIRST 1
6   cell   concentrator_5  FIRST 1
7   cell   concentrator_6  FIRST 1
8   cell   concentrator_7  FIRST 1
9   cell   ict  FIRST 1

cell   pf_3   pf_logic
1   cell   concentrator_0  FIRST 1
2   cell   concentrator_1  FIRST 1
3   cell   concentrator_2  FIRST 1
4   cell   concentrator_3  FIRST 1
5   cell   concentrator_4  FIRST 1
6   cell   concentrator_5  FIRST 1
7   cell   concentrator_6  FIRST 1
8   cell   concentrator_7  FIRST 1
9   cell   ict  FIRST 1

cell   pf_4   pf_logic
1   cell   concentrator_0  FIRST 1
2   cell   concentrator_1  FIRST 1
3   cell   concentrator_2  FIRST 1
4   cell   concentrator_3  FIRST 1
5   cell   concentrator_4  FIRST 1
6   cell   concentrator_5  FIRST 1
7   cell   concentrator_6  FIRST 1
8   cell   concentrator_7  FIRST 1
9   cell   ict  FIRST 1

cell   pf_5   pf_logic
1   cell   concentrator_0  FIRST 1
2   cell   concentrator_1  FIRST 1
3   cell   concentrator_2  FIRST 1
4   cell   concentrator_3  FIRST 1
5   cell   concentrator_4  FIRST 1
6   cell   concentrator_5  FIRST 1
7   cell   concentrator_6  FIRST 1
8   cell   concentrator_7  FIRST 1
9   cell   ict  FIRST 1

cell   pf_6   pf_logic
1   cell   concentrator_0  FIRST 1
2   cell   concentrator_1  FIRST 1
3   cell   concentrator_2  FIRST 1
4   cell   concentrator_3  FIRST 1
5   cell   concentrator_4  FIRST 1
6   cell   concentrator_5  FIRST 1
7   cell   concentrator_6  FIRST 1
8   cell   concentrator_7  FIRST 1
<table>
<thead>
<tr>
<th>cell</th>
<th>pf_7</th>
<th>pf_logic</th>
<th>9</th>
<th>cell</th>
<th>ict</th>
<th>FIRST 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell</td>
<td>concentrator_0</td>
<td>concentrator_logic</td>
<td>1</td>
<td>cell</td>
<td>buffer_0</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>concentrator_1</td>
<td>concentrator_logic</td>
<td>2</td>
<td>cell</td>
<td>boc</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>concentrator_2</td>
<td>concentrator_logic</td>
<td>3</td>
<td>cell</td>
<td>crct</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>concentrator_3</td>
<td>concentrator_logic</td>
<td>4</td>
<td>cell</td>
<td>buffer_1</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>concentrator_4</td>
<td>concentrator_logic</td>
<td>5</td>
<td>cell</td>
<td>boc</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>concentrator_5</td>
<td>concentrator_logic</td>
<td>6</td>
<td>cell</td>
<td>crct</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>concentrator_6</td>
<td>concentrator_logic</td>
<td>7</td>
<td>cell</td>
<td>buffer_2</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>concentrator_7</td>
<td>concentrator_logic</td>
<td>8</td>
<td>cell</td>
<td>boc</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>concentrator_8</td>
<td>concentrator_logic</td>
<td>9</td>
<td>cell</td>
<td>crct</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>clock_tick</td>
<td>clk_cntrl</td>
<td>clock_control_logic</td>
<td>1</td>
<td>clock_tick</td>
<td>EXIT</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>concentrator_0</td>
<td>concentrator_logic</td>
<td>1</td>
<td>cell</td>
<td>buffer_0</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>concentrator_1</td>
<td>concentrator_logic</td>
<td>2</td>
<td>cell</td>
<td>boc</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>concentrator_2</td>
<td>concentrator_logic</td>
<td>3</td>
<td>cell</td>
<td>crct</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>concentrator_3</td>
<td>concentrator_logic</td>
<td>4</td>
<td>cell</td>
<td>buffer_1</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>concentrator_4</td>
<td>concentrator_logic</td>
<td>5</td>
<td>cell</td>
<td>boc</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>concentrator_5</td>
<td>concentrator_logic</td>
<td>6</td>
<td>cell</td>
<td>crct</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>concentrator_6</td>
<td>concentrator_logic</td>
<td>7</td>
<td>cell</td>
<td>buffer_2</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>concentrator_7</td>
<td>concentrator_logic</td>
<td>8</td>
<td>cell</td>
<td>boc</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>concentrator_8</td>
<td>concentrator_logic</td>
<td>9</td>
<td>cell</td>
<td>crct</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>ict</td>
<td></td>
<td></td>
<td>cell</td>
<td>EXIT</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>crct</td>
<td></td>
<td></td>
<td>cell</td>
<td>EXIT</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>boc</td>
<td></td>
<td></td>
<td>cell</td>
<td>EXIT</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>buffer_0</td>
<td>buffer_logic</td>
<td></td>
<td>cell</td>
<td>output_port_0</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>buffer_1</td>
<td>buffer_logic</td>
<td></td>
<td>cell</td>
<td>output_port_1</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>buffer_2</td>
<td>buffer_logic</td>
<td></td>
<td>cell</td>
<td>output_port_2</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>buffer_3</td>
<td>buffer_logic</td>
<td></td>
<td>cell</td>
<td>output_port_3</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>buffer_4</td>
<td>buffer_logic</td>
<td></td>
<td>cell</td>
<td>output_port_4</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>buffer_5</td>
<td>buffer_logic</td>
<td></td>
<td>cell</td>
<td>output_port_5</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>buffer_6</td>
<td>buffer_logic</td>
<td></td>
<td>cell</td>
<td>output_port_6</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>buffer_7</td>
<td>buffer_logic</td>
<td></td>
<td>cell</td>
<td>output_port_7</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>output_port_0</td>
<td>output_logic</td>
<td></td>
<td>cell</td>
<td>EXIT</td>
<td>FIRST 1</td>
</tr>
</tbody>
</table>
cell output_port_1 output_logic 1 cell EXIT FIRST 1
cell output_port_2 output_logic 1 cell EXIT FIRST 1
cell output_port_3 output_logic 1 cell EXIT FIRST 1
cell output_port_4 output_logic 1 cell EXIT FIRST 1
cell output_port_5 output_logic 1 cell EXIT FIRST 1
cell output_port_6 output_logic 1 cell EXIT FIRST 1
cell output_port_7 output_logic 1 cell EXIT FIRST 1

******************************************************************************

Arrivals
******************************************************************************

<table>
<thead>
<tr>
<th>Entity</th>
<th>Location</th>
<th>Qty each</th>
<th>First Time</th>
<th>Occurrences</th>
<th>Frequency</th>
<th>Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock_tick</td>
<td>clk_cntrl</td>
<td>1</td>
<td>0</td>
<td>INF</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>input_port_0</td>
<td>1</td>
<td>0</td>
<td>INF</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>input_port_1</td>
<td>1</td>
<td>0</td>
<td>INF</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>input_port_2</td>
<td>1</td>
<td>0</td>
<td>INF</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>input_port_3</td>
<td>1</td>
<td>0</td>
<td>INF</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>input_port_4</td>
<td>1</td>
<td>0</td>
<td>INF</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>input_port_5</td>
<td>1</td>
<td>0</td>
<td>INF</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>input_port_6</td>
<td>1</td>
<td>0</td>
<td>INF</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>input_port_7</td>
<td>1</td>
<td>0</td>
<td>INF</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

******************************************************************************

Attributes
******************************************************************************

<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>time_in</td>
<td>Real</td>
<td>Entity</td>
</tr>
</tbody>
</table>

#0=blank cell, 1=real cell

activity Integer Entity

#Specifies the address of the output port the cell is to be delivered to
destination_addr Integer Entity

******************************************************************************

Variables [global]
******************************************************************************

<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
<th>Initial value</th>
<th>Stats</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell_delay</td>
<td>Real</td>
<td>0</td>
<td>Basic</td>
</tr>
<tr>
<td>burst_length</td>
<td>Integer</td>
<td>0</td>
<td>Basic</td>
</tr>
<tr>
<td>burst_separation</td>
<td>Integer</td>
<td>0</td>
<td>Basic</td>
</tr>
</tbody>
</table>

******************************************************************************

Arrays
******************************************************************************
ID Dimensions Type
---------- ----------- ----------
conc_array 8,2 Integer
burst_array 8,3 Integer

******************************************************************************
* Macros                                                                       *
******************************************************************************

ID Text
----------
mean_length 5
mean_separation 10
buffer_depth 4
concentrator_output_lines 5
b_size 20
stop_sim_count 12500
input_logic IF ( burst_array[_port, length] > 0 ) THEN
( activity = 1
  time_in = CLOCK()
  destination_addr = burst_array[_port, destination]
  DEC burst_array[_port, length]
 ) ELSE
( burst_separation = P(mean_separation)
  WAIT burst_separation SEC

  burst_array[_port, length] = P(mean_length)
  burst_length = burst_array[_port, length]

  IF ( burst_array[_port, length] > 0 ) THEN
    burst_array[_port, destination] = random_destination_addr(1)
    destination_addr = burst_array[_port, destination]
    DEC burst_array[_port, length]
    activity = 1
    time_in = CLOCK()
  ) ELSE
    activity = 0
)

pf_logic IF (activity = 1) THEN
  ROUTE (destination_addr + 1)
ELSE
  ROUTE 9

concentrator_logic INC _ccc
IF ( _ccc <= concentrator_output_lines) THEN
(}
IF ( CONTENTS ( _bln ) < b_size ) THEN
  ROUTE 1
ELSE
  ROUTE 2
}
ELSE
  ROUTE 3
  WAIT 1 SEC

output_logic
  IF (destination_addr <> (LOCATION() - output_port_base_addr)) THEN
    DISPLAY "A cell destined for output port " $ destination_addr $ ' arrived at port ' $ LOCATION()
  - output_port_base_addr

buffer_logic
  cell_delay = CLOCK() - time_in
  WAIT 1 SEC
  INT i = 1
  WHILE (i <= 8) DO
    { conc_array[i, ccc] = 0
      INC i
    }
  IF (ENTRIES() > stop_sim_count) THEN
    STOP
  initialization_logic
  INT i = 0
  WHILE (i <= 8) DO
    { conc_array[i + 1, bln] = buffer_base_addr + (concentrator_output_lines + 1) * i
      conc_array[i + 1, ccc] = 0
      burst_array[i + 1, length] = 0
      burst_array[i + 1, separation] = 0
      burst_array[i + 1, destination] = 0
      INC i
    }

buffer_base_addr 28
conc_base_addr 16
output_port_base_addr 76
true 1
false 0
length 1
separation 2
destination 3
bln 1
ccc 2
_conc_array[LOCATION()] - conc_base_addr, ccc
_LOC ( conc_array[LOCATION()] - conc_base_addr, bln )
_port LOCATION()
<table>
<thead>
<tr>
<th>ID</th>
<th>random_destination_add</th>
<th>Type</th>
<th>Cumulative</th>
<th>Percentage</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>12.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>12.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td>12.5</td>
</tr>
</tbody>
</table>

Stream # | Seed # | React | Streams
---|--------|-------|--------
1 | 73 | No |
Appendix B  Sunshine Switch Bursty Model Text
## Model Notes:

Sunshine Model S7 (Bursty)

A simulation version of the sunshine ATM switching architecture.

**No. Banyans:** 4

**Buffer Depth:** 5

**Buffer arrangement:** 1 recirculator/20 buffers per output port

**Traffic pattern input:** Bursty

**Time Units:** Seconds

**Distance Units:** Feet

**Initialization Logic:** initialization_logic

### Locations

<table>
<thead>
<tr>
<th>Name</th>
<th>Cap</th>
<th>Units</th>
<th>Stats</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>input_port_0</td>
<td>1</td>
<td>1</td>
<td>None</td>
<td>Oldest,</td>
</tr>
<tr>
<td>input_port_1</td>
<td>1</td>
<td>1</td>
<td>None</td>
<td>Oldest,</td>
</tr>
<tr>
<td>input_port_2</td>
<td>1</td>
<td>1</td>
<td>None</td>
<td>Oldest,</td>
</tr>
<tr>
<td>input_port_3</td>
<td>1</td>
<td>1</td>
<td>None</td>
<td>Oldest,</td>
</tr>
<tr>
<td>input_port_4</td>
<td>1</td>
<td>1</td>
<td>None</td>
<td>Oldest,</td>
</tr>
<tr>
<td>input_port_5</td>
<td>1</td>
<td>1</td>
<td>None</td>
<td>Oldest,</td>
</tr>
<tr>
<td>input_port_6</td>
<td>1</td>
<td>1</td>
<td>None</td>
<td>Oldest,</td>
</tr>
<tr>
<td>input_port_7</td>
<td>1</td>
<td>1</td>
<td>None</td>
<td>Oldest,</td>
</tr>
<tr>
<td>batcher_trap</td>
<td>1</td>
<td>1</td>
<td>Basic</td>
<td>Min{adjusted_priority},</td>
</tr>
<tr>
<td>recirculator</td>
<td>INFINITE</td>
<td>1</td>
<td>Basic</td>
<td>Oldest,</td>
</tr>
<tr>
<td>recirculator_trap</td>
<td>INFINITE</td>
<td>1</td>
<td>Basic</td>
<td>Oldest,</td>
</tr>
<tr>
<td>recirculated_cells</td>
<td>INFINITE</td>
<td>1</td>
<td>Basic</td>
<td>Oldest,</td>
</tr>
<tr>
<td>banyan_router</td>
<td>15</td>
<td>1</td>
<td>Basic</td>
<td>Oldest,</td>
</tr>
<tr>
<td>inactive_cell_trap</td>
<td>INFINITE</td>
<td>1</td>
<td>Basic</td>
<td>Oldest,</td>
</tr>
</tbody>
</table>
### Buffer Overflow Cell Trap

<p>| boot     | buffer_0 | buffer_0_1 | buffer_0_2 | buffer_0_3 | buffer_0_4 | buffer_1 | buffer_1_1 | buffer_1_2 | buffer_1_3 | buffer_1_4 | buffer_2 | buffer_2_1 | buffer_2_2 | buffer_2_3 | buffer_2_4 | buffer_3 | buffer_3_1 | buffer_3_2 | buffer_3_3 | buffer_3_4 | buffer_4 | buffer_4_1 | buffer_4_2 | buffer_4_3 | buffer_4_4 | buffer_5 | buffer_5_1 | buffer_5_2 | buffer_5_3 | buffer_5_4 | buffer_6 | buffer_6_1 | buffer_6_2 | buffer_6_3 | buffer_6_4 | buffer_7 | buffer_7_1 | buffer_7_2 | buffer_7_3 | buffer_7_4 | output_port_0 | output_port_1 | output_port_2 | output_port_3 | output_port_4 | output_port_5 | output_port_6 | output_port_7 | clk_cntrl |
|----------|----------|------------|------------|------------|------------|----------|------------|------------|------------|------------|----------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|-----------|
| INFINITE| 100      | 100        | 100        | 100        | 100        | 100      | 100         | 100         | 100         | 100         | 100      | 100        | 100        | 100        | 100        | 100        | 100        | 100        | 100        | 100        | 100        | 100        | 100        | 100        | 100        | 100        | 100        | 100        | 100        | 100        | 100       |
| Basic   | Basic    | Basic      | Basic      | Basic      | Basic      | Basic    | Basic       | None       | Basic      | None       | Basic    | None       | None       | Basic      | None       | Basic      | None       | Basic     | None       | None       | Basic    | None       | None       | Basic      | None       | Basic      | None       | Basic     | None       | None       | None       | None       | 2         | 1         |
| Oldest, | FIFO,    | Oldest,    | Oldest,    | FIFO,      | FIFO,      | FIFO,    | FIFO        | FIFO       | FIFO       | FIFO       | FIFO,    | FIFO       | FIFO       | FIFO       | FIFO,     | FIFO       | FIFO       | FIFO     | FIFO       | FIFO       | FIFO       | FIFO       | FIFO       | FIFO       | FIFO       | FIFO       | FIFO       | FIFO       | FIFO       | FIFO       | Basic     | 1         |
| By turn |         |            |            |            |            |         |             |            |            |            |         |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |          |          |</p>
<table>
<thead>
<tr>
<th>Entity</th>
<th>Location</th>
<th>Operation</th>
<th>Blk</th>
<th>Output</th>
<th>Destination</th>
<th>Rule</th>
<th>Move Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell</td>
<td>input_port_0</td>
<td>input_logic</td>
<td>1</td>
<td>cell</td>
<td>batcher_trap</td>
<td>FIRST 1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>input_port_1</td>
<td>input_logic</td>
<td>2</td>
<td>cell</td>
<td>EXIT</td>
<td>FIRST 1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>input_port_2</td>
<td>input_logic</td>
<td>1</td>
<td>cell</td>
<td>batcher_trap</td>
<td>FIRST 1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>input_port_3</td>
<td>input_logic</td>
<td>2</td>
<td>cell</td>
<td>EXIT</td>
<td>FIRST 1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>input_port_4</td>
<td>input_logic</td>
<td>1</td>
<td>cell</td>
<td>batcher_trap</td>
<td>FIRST 1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>input_port_5</td>
<td>input_logic</td>
<td>2</td>
<td>cell</td>
<td>EXIT</td>
<td>FIRST 1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>input_port_6</td>
<td>input_logic</td>
<td>1</td>
<td>cell</td>
<td>batcher_trap</td>
<td>FIRST 1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>input_port_7</td>
<td>input_logic</td>
<td>2</td>
<td>cell</td>
<td>EXIT</td>
<td>FIRST 1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>batcher_trap</td>
<td>batcher_trap_logic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hold_cell</td>
<td>batcher_trap</td>
<td>hold_cell_logic</td>
<td></td>
<td>hold_cell</td>
<td>EXIT</td>
<td>FIRST 1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>recirculator</td>
<td>recirculator_logic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>recirculated_cell_port</td>
<td>recirculated_cell_port_logic</td>
<td>1</td>
<td>cell</td>
<td>batcher_trap</td>
<td>FIRST 1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>banyan_router</td>
<td>banyan_router_logic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>root</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>ixt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>boot</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>buffer_0</td>
<td>buffer_logic</td>
<td>1</td>
<td>cell</td>
<td>output_port_0</td>
<td>FIRST 1</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>buffer_1</td>
<td>buffer_logic</td>
<td>1</td>
<td>cell</td>
<td>output_port_1</td>
<td>FIRST 1</td>
<td></td>
</tr>
</tbody>
</table>
cell buffer_2 buffer_logic 1 cell output_port_2 FIRST 1
cell buffer_3 buffer_logic 1 cell output_port_3 FIRST 1
cell buffer_4 buffer_logic 1 cell output_port_4 FIRST 1
cell buffer_5 buffer_logic 1 cell output_port_5 FIRST 1
cell buffer_6 buffer_logic 1 cell output_port_6 FIRST 1
cell buffer_7 buffer_logic 1 cell output_port_7 FIRST 1
cell output_port_0 output_logic 1 cell EXIT FIRST 1
cell output_port_1 output_logic 1 cell EXIT FIRST 1
cell output_port_2 output_logic 1 cell EXIT FIRST 1
cell output_port_3 output_logic 1 cell EXIT FIRST 1
cell output_port_4 output_logic 1 cell EXIT FIRST 1
cell output_port_5 output_logic 1 cell EXIT FIRST 1
cell output_port_6 output_logic 1 cell EXIT FIRST 1
cell output_port_7 output_logic 1 cell EXIT FIRST 1
clock_tick clk_ctrl clk_ctrl_logic 1 clock_tick EXIT FIRST 1

******************************************************************************

。 Arrivals 。

<table>
<thead>
<tr>
<th>Entity</th>
<th>Location</th>
<th>Qty each</th>
<th>First Time Occurrences</th>
<th>Frequency</th>
<th>Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>hold_cell</td>
<td>batcher_trap</td>
<td>1</td>
<td>0</td>
<td>INF</td>
<td>1</td>
</tr>
<tr>
<td>cell</td>
<td>input_port_0</td>
<td>1</td>
<td>0</td>
<td>INF</td>
<td>1</td>
</tr>
<tr>
<td>cell</td>
<td>input_port_1</td>
<td>1</td>
<td>0</td>
<td>INF</td>
<td>1</td>
</tr>
<tr>
<td>cell</td>
<td>input_port_2</td>
<td>1</td>
<td>0</td>
<td>INF</td>
<td>1</td>
</tr>
<tr>
<td>cell</td>
<td>input_port_3</td>
<td>1</td>
<td>0</td>
<td>INF</td>
<td>1</td>
</tr>
<tr>
<td>cell</td>
<td>input_port_4</td>
<td>1</td>
<td>0</td>
<td>INF</td>
<td>1</td>
</tr>
<tr>
<td>cell</td>
<td>input_port_5</td>
<td>1</td>
<td>0</td>
<td>INF</td>
<td>1</td>
</tr>
<tr>
<td>cell</td>
<td>input_port_6</td>
<td>1</td>
<td>0</td>
<td>INF</td>
<td>1</td>
</tr>
<tr>
<td>cell</td>
<td>input_port_7</td>
<td>1</td>
<td>0</td>
<td>INF</td>
<td>1</td>
</tr>
<tr>
<td>clock_tick</td>
<td>clk_ctrl</td>
<td>1</td>
<td>0</td>
<td>INF</td>
<td>1</td>
</tr>
</tbody>
</table>

******************************************************************************

。 Attributes 。

<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>time_in</td>
<td>Real</td>
<td>Entity</td>
</tr>
<tr>
<td>activity</td>
<td>Integer</td>
<td>Entity</td>
</tr>
<tr>
<td>destination_addr</td>
<td>Integer</td>
<td>Entity</td>
</tr>
<tr>
<td>service_priority</td>
<td>Integer</td>
<td>Entity</td>
</tr>
<tr>
<td>adjusted_priority</td>
<td>Integer</td>
<td>Entity</td>
</tr>
</tbody>
</table>

******************************************************************************

。 Variables (global) 。

<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
<th>Initial value</th>
<th>Stats</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell_delay</td>
<td>Real</td>
<td>0</td>
<td>Basic</td>
</tr>
<tr>
<td>burst_length</td>
<td>Integer</td>
<td>0</td>
<td>Basic</td>
</tr>
</tbody>
</table>
burst separation Integer 0 Basic
current_addr Integer 99 None
cells_to_same_addr Integer 0 None
release Integer 0 None
recirculators_used Integer 0 None
extra_clock_cells Integer 0 None

******************************************************************************
| Arrays                          |
******************************************************************************

<table>
<thead>
<tr>
<th>ID</th>
<th>Dimensions</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>burst_array</td>
<td>8,3</td>
<td>Integer</td>
</tr>
</tbody>
</table>

******************************************************************************
| Macros                           |
******************************************************************************

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean_length</td>
<td>5</td>
</tr>
<tr>
<td>mean_separation</td>
<td>10</td>
</tr>
<tr>
<td>stop_sim_count</td>
<td>12500</td>
</tr>
<tr>
<td>no_recirculators</td>
<td>4</td>
</tr>
<tr>
<td>k_banyans</td>
<td>5</td>
</tr>
<tr>
<td>buffer_depth</td>
<td>20</td>
</tr>
<tr>
<td>b_size</td>
<td>56</td>
</tr>
<tr>
<td>output_port_base_addr</td>
<td>16</td>
</tr>
<tr>
<td>buffer_base_addr</td>
<td>5</td>
</tr>
<tr>
<td>buffer_spacing</td>
<td>WAIT UNTIL (recirculators_used = 0)</td>
</tr>
<tr>
<td>clk_ctrl_logic</td>
<td>release = true</td>
</tr>
<tr>
<td>hold_cell_logic</td>
<td>WAIT UNTIL (release = true)</td>
</tr>
<tr>
<td>batcher_trap_logic</td>
<td>IF (current_addr = destination_addr) THEN</td>
</tr>
<tr>
<td></td>
<td>IF (cells_to_same_addr &lt;= k_banyans) THEN</td>
</tr>
<tr>
<td></td>
<td>INC cells_to_same_addr</td>
</tr>
<tr>
<td></td>
<td>ELSE</td>
</tr>
<tr>
<td></td>
<td>ROUTE</td>
</tr>
</tbody>
</table>
IF (activity = 1) THEN
  ROUTE 2
ELSE
  ROUTE 1
END

banyan_router_logic

WAIT 1 SEC

IF (activity = 1) THEN
  IF ( CONTENTS (_buffer) < b_size ) THEN
    ROUTE (destination_addr + 1)
  ELSE
    ROUTE 10
  END
ELSE
  ROUTE 9
END

recirculator_logic

IF (recirculators_used < no_recirculators) THEN
  INC recirculators_used
  DEC service_priority
  adjusted_priority = (destination_addr + 10) + service_priority
  ROUTE 1
ELSE
  ROUTE 2
END

recirculated_cell_port_logic

WAIT 1 SEC
DEC recirculators_used

buffer_logic

WAIT 1 SEC

output_logic

WAIT 1 SEC

output_port_base_addr

IF (destination_addr <> (LOCATION() - output_port_base_addr)) THEN
  DISPLAY "A cell destined for output port " $ destination_addr $ 'arrived at port $ (LOCATION()) -

input_logic

cell_delay = CLOCK() - time_in
IF (burst_array[_port, length] > 0) THEN
  activity = 1
  time_in = CLOCK()
  destination_addr = burst_array[_port, destination]
  DEC burst_array[_port, length]
END
ELSE

burst_separation = P(mean_separation)
wait burst_separation SEC

burst_array[.port, .length] = P(mean_length)
burst_length = burst_array[.port, .length]

if (burst_array[.port, .length] > 0) then
  burst_array[.port, destination] = random_destination_addr(1)
destination_addr = burst_array[.port, destination]
DEC burst_array[.port, .length]
activity = 1
  time_in = CLOCK()
else
  activity = 0
endif

service_priority = 9
adjusted_priority = (destination_addr * 10) + service_priority

initialization_logic
int i = 0
while (i < 8) do
  burst_array[i + 1, .length] = 0
  burst_array[i + 1, separation] = 0
  burst_array[i + 1, destination] = 0
  inc i
endwhile

_buffer
loc (destination_addr * buffer_spacing) + buffer_base_addr

_port
location()
length 1
separation 2
destination 3
true 1
false 0

*******************************************************************************
* User Distributions
*******************************************************************************

<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
<th>Cumulative</th>
<th>Percentage</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>random_destination_addr</td>
<td>Discrete</td>
<td>No</td>
<td>12.5</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>12.5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>12.5</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>12.5</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>12.5</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>12.5</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>12.5</td>
<td>6</td>
</tr>
</tbody>
</table>
* * *

Formatted Listing of Model: *
X:\PM3\MODELS\HELIX\HB2.MOD *

* * *

-------------------------------
| Time Units: | Seconds |
| Distance Units: | Feet |
| Initialization Logic: | initialization_logic |

-------------------------------

<table>
<thead>
<tr>
<th>Name</th>
<th>Cap</th>
<th>Units</th>
<th>Stats</th>
<th>Rules</th>
</tr>
</thead>
</table>

# Broadcast unit stage 1, ports 0 - 7

bu_s1n0 2 1 None Oldest, ,
bu_s1n1 2 1 None Oldest, ,
bu_s1n2 2 1 None Oldest, ,
bu_s1n3 2 1 None Oldest, ,
bu_s1n4 2 1 None Oldest, ,
bu_s1n5 2 1 None Oldest, ,
bu_s1n6 2 1 None Oldest, ,
bu_s1n7 2 1 None Oldest, ,

# Broadcast unit stage 2, ports 0 - 7

bu_s2n0 2 1 None Oldest, ,
bu_s2n1 2 1 None Oldest, ,
bu_s2n2 2 1 None Oldest, ,
bu_s2n3 2 1 None Oldest, ,
bu_s2n4 2 1 None Oldest, ,
bu_s2n5 2 1 None Oldest, ,
bu_s2n6 2 1 None Oldest, ,
bu_s2n7 2 1 None Oldest, ,

# Broadcast unit stage 3, ports 0 - 7

bu_s3n0 2 1 None Oldest, ,
bu_s3n1 2 1 None Oldest, ,
bu_s3n2 2 1 None Oldest, ,
bu_s3n3 2 1 None Oldest, ,
bu_s3n4 2 1 None Oldest, ,
bu_s3n5 2 1 None Oldest, ,
bu_s3n6 2 1 None Oldest, ,
bu_s3n7 2 1 None Oldest, ,

# Buffer unit stage 1, W's 0 - 15

b_s1n0 100 1 Basic Oldest, FIFO,
b_s1n1 100 1 Basic Oldest, FIFO,
b_s1n2 100 1 Basic Oldest, FIFO,
b_s1n3 100 1 Basic Oldest, FIFO,
b_s1n4 100 1 Basic Oldest, FIFO,
b_s1n5 100 1 Basic Oldest, FIFO,
b_s1n6  100  1  Basic  Oldest, FIFO,
b_s1n7  100  1  Basic  Oldest, FIFO,
b_s1n8  100  1  Basic  Oldest, FIFO,
b_s1n9  100  1  Basic  Oldest, FIFO,
b_s1n10  100  1  Basic  Oldest, FIFO,
b_s1n11  100  1  Basic  Oldest, FIFO,
b_s1n12  100  1  Basic  Oldest, FIFO,
b_s1n13  100  1  Basic  Oldest, FIFO,
b_s1n14  100  1  Basic  Oldest, FIFO,
b_s1n15  100  1  Basic  Oldest, FIFO,

# Buffer unit stage 2, #s. 0 - 15
b_s2n0  100  1  Basic  Oldest, FIFO,
b_s2n1  100  1  Basic  Oldest, FIFO,
b_s2n2  100  1  Basic  Oldest, FIFO,
b_s2n3  100  1  Basic  Oldest, FIFO,
b_s2n4  100  1  Basic  Oldest, FIFO,
b_s2n5  100  1  Basic  Oldest, FIFO,
b_s2n6  100  1  Basic  Oldest, FIFO,
b_s2n7  100  1  Basic  Oldest, FIFO,
b_s2n8  100  1  Basic  Oldest, FIFO,
b_s2n9  100  1  Basic  Oldest, FIFO,
b_s2n10  100  1  Basic  Oldest, FIFO,
b_s2n11  100  1  Basic  Oldest, FIFO,
b_s2n12  100  1  Basic  Oldest, FIFO,
b_s2n13  100  1  Basic  Oldest, FIFO,
b_s2n14  100  1  Basic  Oldest, FIFO,
b_s2n15  100  1  Basic  Oldest, FIFO,

# Buffer unit stage 3, #s. 0 - 15
b_s3n0  100  1  Basic  Oldest, FIFO,
b_s3n1  100  1  Basic  Oldest, FIFO,
b_s3n2  100  1  Basic  Oldest, FIFO,
b_s3n3  100  1  Basic  Oldest, FIFO,
b_s3n4  100  1  Basic  Oldest, FIFO,
b_s3n5  100  1  Basic  Oldest, FIFO,
b_s3n6  100  1  Basic  Oldest, FIFO,
b_s3n7  100  1  Basic  Oldest, FIFO,
b_s3n8  100  1  Basic  Oldest, FIFO,
b_s3n9  100  1  Basic  Oldest, FIFO,
b_s3n10  100  1  Basic  Oldest, FIFO,
b_s3n11  100  1  Basic  Oldest, FIFO,
b_s3n12  100  1  Basic  Oldest, FIFO,
b_s3n13  100  1  Basic  Oldest, FIFO,
b_s3n14  100  1  Basic  Oldest, FIFO,
b_s3n15  100  1  Basic  Oldest, FIFO,

# Concentrator Control
z_cmtr1  1  1  Basic  Oldest, ,

# Concentrator unit stage 1, no. 0
concs1n0  15  1  None  Oldest, ,

# Concentrator unit stage 1, no. 1
concs1n1  15  1  None  Oldest, ,
# Concentrator unit stage 2, no. 0
conc_s2n0 15 1 None Oldest,
#
# Concentrator unit stage 2, no. 1
conc_s2n1 15 1 None Oldest,
#
# Concentrator unit stage 2, no. 2
conc_s2n2 15 1 None Oldest,
#
# Concentrator unit stage 2, no. 3
conc_s2n3 15 1 None Oldest,
#
# Concentrator unit stage 3, no. 0
conc_s3n0 15 1 None Oldest,
#
# Concentrator unit stage 3, no. 1
conc_s3n1 15 1 None Oldest,
#
# Concentrator unit stage 3, no. 2
conc_s3n2 15 1 None Oldest,
#
# Concentrator unit stage 3, no. 3
conc_s3n3 15 1 None Oldest,
#
# Concentrator unit stage 3, no. 4
conc_s3n4 15 1 None Oldest,
#
# Concentrator unit stage 3, no. 5
conc_s3n5 15 1 None Oldest,
#
# Concentrator unit stage 3, no. 6
conc_s3n6 15 1 None Oldest,
#
# Concentrator unit stage 3, no. 7
conc_s3n7 15 1 None Oldest,
#
# Dummy Cell Traps
dct_1 INFINITY 1 Basic Oldest,
dct_2 INFINITY 1 Basic Oldest,
dct_3 INFINITY 1 Basic Oldest,
#
# Broadcast Unit Overflow Traps
buot_1 INFINITY 1 Basic Oldest,
buot_2 INFINITY 1 Basic Oldest,
buot_3 INFINITY 1 Basic Oldest,
#
# Model input ports
input_port_0 1 1 None Oldest,
input_port_1 1 1 None Oldest,
input_port_2 1 1 None Oldest,
input_port_3 1 1 None Oldest,
input_port_4 1 1 None Oldest,
input_port_5 1 1 None Oldest,
input_port_6 1 1 None Oldest,
input_port_7 1 1 None Oldest,
# Model output ports

<table>
<thead>
<tr>
<th>output_port_0</th>
<th>1</th>
<th>Basic</th>
<th>Oldest,</th>
</tr>
</thead>
<tbody>
<tr>
<td>output_port_1</td>
<td>1</td>
<td>Basic</td>
<td>Oldest,</td>
</tr>
<tr>
<td>output_port_2</td>
<td>1</td>
<td>Basic</td>
<td>Oldest,</td>
</tr>
<tr>
<td>output_port_3</td>
<td>1</td>
<td>Basic</td>
<td>Oldest,</td>
</tr>
<tr>
<td>output_port_4</td>
<td>1</td>
<td>Basic</td>
<td>Oldest,</td>
</tr>
<tr>
<td>output_port_5</td>
<td>1</td>
<td>Basic</td>
<td>Oldest,</td>
</tr>
<tr>
<td>output_port_6</td>
<td>1</td>
<td>Basic</td>
<td>Oldest,</td>
</tr>
<tr>
<td>output_port_7</td>
<td>1</td>
<td>Basic</td>
<td>Oldest,</td>
</tr>
</tbody>
</table>

**Entities**

<table>
<thead>
<tr>
<th>Name</th>
<th>Speed (fpm)</th>
<th>Stats</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell_b0</td>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>cell_b1</td>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>cell_b2</td>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>cell_b3</td>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>cell_b4</td>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>cell_b5</td>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>cell_b6</td>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>cell_b7</td>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>cell</td>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>clock_tick</td>
<td>0</td>
<td>None</td>
</tr>
</tbody>
</table>

**Processing**

<table>
<thead>
<tr>
<th>Entity</th>
<th>Location</th>
<th>Operation</th>
<th>Blk Output</th>
<th>Destination</th>
<th>Rule</th>
<th>Move Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell</td>
<td>input_port_0</td>
<td>input_logic</td>
<td>1</td>
<td>cell</td>
<td>bu_sin0</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>input_port_1</td>
<td>input_logic</td>
<td>2</td>
<td>cell</td>
<td>EXIT</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>input_port_2</td>
<td>input_logic</td>
<td>1</td>
<td>cell</td>
<td>bu_sin1</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>input_port_3</td>
<td>input_logic</td>
<td>2</td>
<td>cell</td>
<td>EXIT</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>input_port_4</td>
<td>input_logic</td>
<td>1</td>
<td>cell</td>
<td>bu_sin2</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>input_port_5</td>
<td>input_logic</td>
<td>2</td>
<td>cell</td>
<td>EXIT</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>input_port_6</td>
<td>input_logic</td>
<td>1</td>
<td>cell</td>
<td>bu_sin3</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>input_port_7</td>
<td>input_logic</td>
<td>2</td>
<td>cell</td>
<td>EXIT</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>bu_sin5</td>
<td>bu_sl_logic</td>
<td>1</td>
<td>cell</td>
<td>bu_sin6</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>bu_sin7</td>
<td>bu_sl_logic</td>
<td>2</td>
<td>cell</td>
<td>EXIT</td>
<td>FIRST 1</td>
</tr>
<tr>
<td>cell</td>
<td>bu_s1n1</td>
<td>bu_s1_logic</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>cell</td>
<td>b_s1n8</td>
<td>FIRST 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>cell</td>
<td>buot_1</td>
<td>FIRST 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>cell</td>
<td>dct_1</td>
<td>FIRST 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>cell</td>
<td>b_s1n1</td>
<td>FIRST 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>cell</td>
<td>b_s1n9</td>
<td>FIRST 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>cell</td>
<td>buot_1</td>
<td>FIRST 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>cell</td>
<td>dct_1</td>
<td>FIRST 1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cell</th>
<th>bu_s1n2</th>
<th>bu_s1_logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>cell</td>
<td>b_s1n2</td>
</tr>
<tr>
<td>2</td>
<td>cell</td>
<td>b_s1n10</td>
</tr>
<tr>
<td>3</td>
<td>cell</td>
<td>buot_1</td>
</tr>
<tr>
<td>4</td>
<td>cell</td>
<td>dct_1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cell</th>
<th>bu_s1n3</th>
<th>bu_s1_logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>cell</td>
<td>b_s1n3</td>
</tr>
<tr>
<td>2</td>
<td>cell</td>
<td>b_s1n11</td>
</tr>
<tr>
<td>3</td>
<td>cell</td>
<td>buot_1</td>
</tr>
<tr>
<td>4</td>
<td>cell</td>
<td>dct_1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cell</th>
<th>bu_s1n4</th>
<th>bu_s1_logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>cell</td>
<td>b_s1n4</td>
</tr>
<tr>
<td>2</td>
<td>cell</td>
<td>b_s1n12</td>
</tr>
<tr>
<td>3</td>
<td>cell</td>
<td>buot_1</td>
</tr>
<tr>
<td>4</td>
<td>cell</td>
<td>dct_1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cell</th>
<th>bu_s1n5</th>
<th>bu_s1_logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>cell</td>
<td>b_s1n5</td>
</tr>
<tr>
<td>2</td>
<td>cell</td>
<td>b_s1n13</td>
</tr>
<tr>
<td>3</td>
<td>cell</td>
<td>buot_1</td>
</tr>
<tr>
<td>4</td>
<td>cell</td>
<td>dct_1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cell</th>
<th>bu_s1n6</th>
<th>bu_s1_logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>cell</td>
<td>b_s1n6</td>
</tr>
<tr>
<td>2</td>
<td>cell</td>
<td>b_s1n14</td>
</tr>
<tr>
<td>3</td>
<td>cell</td>
<td>buot_1</td>
</tr>
<tr>
<td>4</td>
<td>cell</td>
<td>dct_1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cell</th>
<th>bu_s1n7</th>
<th>bu_s1_logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>cell</td>
<td>b_s1n7</td>
</tr>
<tr>
<td>2</td>
<td>cell</td>
<td>b_s1n15</td>
</tr>
<tr>
<td>3</td>
<td>cell</td>
<td>buot_1</td>
</tr>
<tr>
<td>4</td>
<td>cell</td>
<td>dct_1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cell</th>
<th>bu_s2n0</th>
<th>bu_s2_logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>cell</td>
<td>b_s2n0</td>
</tr>
<tr>
<td>2</td>
<td>cell</td>
<td>b_s2n4</td>
</tr>
<tr>
<td>3</td>
<td>cell</td>
<td>buot_2</td>
</tr>
<tr>
<td>4</td>
<td>cell</td>
<td>dct_2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cell</th>
<th>bu_s2n1</th>
<th>bu_s2_logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>cell</td>
<td>b_s2n1</td>
</tr>
<tr>
<td>2</td>
<td>cell</td>
<td>b_s2n5</td>
</tr>
<tr>
<td>3</td>
<td>cell</td>
<td>buot_2</td>
</tr>
<tr>
<td>4</td>
<td>cell</td>
<td>dct_2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cell</th>
<th>bu_s2n2</th>
<th>bu_s2_logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>cell</td>
<td>b_s2n2</td>
</tr>
<tr>
<td>2</td>
<td>cell</td>
<td>b_s2n6</td>
</tr>
<tr>
<td>3</td>
<td>cell</td>
<td>buot_2</td>
</tr>
<tr>
<td>4</td>
<td>cell</td>
<td>dct_2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cell</th>
<th>bu_s2n3</th>
<th>bu_s2_logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>cell</td>
<td>b_s2n3</td>
</tr>
<tr>
<td>2</td>
<td>cell</td>
<td>b_s2n7</td>
</tr>
<tr>
<td>3</td>
<td>cell</td>
<td>buot_2</td>
</tr>
<tr>
<td>4</td>
<td>cell</td>
<td>dct_2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cell</th>
<th>bu_s2n4</th>
<th>bu_s2_logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>cell</td>
<td>b_s2n8</td>
</tr>
<tr>
<td>2</td>
<td>cell</td>
<td>b_s2n12</td>
</tr>
<tr>
<td>3</td>
<td>cell</td>
<td>buot_2</td>
</tr>
<tr>
<td>4</td>
<td>cell</td>
<td>dct_2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cell</th>
<th>bu_s2n5</th>
<th>bu_s2_logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>cell</td>
<td>b_s2n9</td>
</tr>
<tr>
<td>2</td>
<td>cell</td>
<td>b_s2n13</td>
</tr>
<tr>
<td>3</td>
<td>cell</td>
<td>buot_2</td>
</tr>
<tr>
<td>4</td>
<td>cell</td>
<td>dct_2</td>
</tr>
<tr>
<td>Entity</td>
<td>Location</td>
<td>Qty each</td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>clock_tick</td>
<td>c_cmtr</td>
<td>1</td>
</tr>
<tr>
<td>cell</td>
<td>input_port_0</td>
<td>1</td>
</tr>
<tr>
<td>cell</td>
<td>input_port_1</td>
<td>1</td>
</tr>
<tr>
<td>cell</td>
<td>input_port_2</td>
<td>1</td>
</tr>
<tr>
<td>cell</td>
<td>input_port_3</td>
<td>1</td>
</tr>
<tr>
<td>cell</td>
<td>input_port_4</td>
<td>1</td>
</tr>
<tr>
<td>cell</td>
<td>input_port_5</td>
<td>1</td>
</tr>
<tr>
<td>cell</td>
<td>input_port_6</td>
<td>1</td>
</tr>
<tr>
<td>cell</td>
<td>input_port_7</td>
<td>1</td>
</tr>
</tbody>
</table>
cell input_port_6 1 0 INF 1
cell input_port_7 1 0 INF 1

*****************************************************************************
*                                                                          *
* Attributes                                                               *
*****************************************************************************

ID         Type         Classification
---------- -----------  -------------------
dummy      Integer      Entity

#0 = real, 1 = dummy cell
#Specifies the address of the output port the cell is to be delivered to
destination_addr Integer Entity
time_in     Real        Entity

*****************************************************************************
*                                                                          *
* Variables [global]                                                       *
*****************************************************************************

ID         Type         Initial value Stats
---------- -----------  -------------------
cell_delay Real         0 Basic
burst_length Integer     0 Basic
burst_separation Integer 0 Basic
c_ctrl_unit Integer     0 None
c_pointer    Integer     1 None

*****************************************************************************
*                                                                          *
* Arrays                                                                  *
*****************************************************************************

ID         Dimensions Type
---------- ----------- -----
c_array    14,12     Integer
bu_s1_array 8,2     Integer
bu_s2_array 8,4     Integer
bu_s3_array 8,4     Integer
burst_array 8,100   Integer

*****************************************************************************
*                                                                          *
* Macros                                                                  *
*****************************************************************************

ID         Text
---------- -----
b_s1_size   6
b_s2_size   6
b_s3_size   12
mean_length 5
mean_separation 10
stop_sim_count 12500
initialization logic INT i

i = 1
WHILE (i <= 14) DO
{
    c_array[i, p1] = 0
    c_array[i, p2] = 0
    c_array[i, p2r] = 0
    c_array[i, p2rv] = 99
    c_array[i, rcf] = 0
    c_array[i, hsa] = true
    c_array[i, rca] = false
    c_array[i, nin] = 73 + i
    INC i
}

c_array[c_s1n0, hol] = c_s1_input_size
        c_array[c_s1n1, hol] = c_s1_input_size
        c_array[c_s2n0, hol] = c_s2_input_size
        c_array[c_s2n1, hol] = c_s2_input_size
        c_array[c_s2n2, hol] = c_s2_input_size
        c_array[c_s2n3, hol] = c_s2_input_size
        c_array[c_s3n0, hol] = c_s3_input_size
        c_array[c_s3n1, hol] = c_s3_input_size
        c_array[c_s3n2, hol] = c_s3_input_size
        c_array[c_s3n3, hol] = c_s3_input_size
        c_array[c_s3n4, hol] = c_s3_input_size
        c_array[c_s3n5, hol] = c_s3_input_size
        c_array[c_s3n6, hol] = c_s3_input_size
        c_array[c_s3n7, hol] = c_s3_input_size

        c_array[c_s1n0, iba] = c_s1n0_base
        c_array[c_s1n1, iba] = c_s1n1_base
        c_array[c_s2n0, iba] = c_s2n0_base
        c_array[c_s2n1, iba] = c_s2n1_base
        c_array[c_s2n2, iba] = c_s2n2_base
        c_array[c_s2n3, iba] = c_s2n3_base
        c_array[c_s3n0, iba] = c_s3n0_base
        c_array[c_s3n1, iba] = c_s3n1_base
        c_array[c_s3n2, iba] = c_s3n2_base
        c_array[c_s3n3, iba] = c_s3n3_base
        c_array[c_s3n4, iba] = c_s3n4_base
        c_array[c_s3n5, iba] = c_s3n5_base
        c_array[c_s3n6, iba] = c_s3n6_base
        c_array[c_s3n7, iba] = c_s3n7_base

        c_array[c_s1n0, cis] = c_s1_input_size
        c_array[c_s1n1, cis] = c_s1_input_size
        c_array[c_s2n0, cis] = c_s2_input_size
        c_array[c_s2n1, cis] = c_s2_input_size
        c_array[c_s2n2, cis] = c_s2_input_size
        c_array[c_s2n3, cis] = c_s2_input_size
        c_array[c_s3n0, cis] = c_s3_input_size
        c_array[c_s3n1, cis] = c_s3_input_size
        c_array[c_s3n2, cis] = c_s3_input_size
c_array[c_s3n3, cis] = c_s3_input_size

c_array[c_s3n4, cis] = c_s3_input_size

c_array[c_s3n5, cis] = c_s3_input_size

c_array[c_s3n6, cis] = c_s3_input_size

c_array[c_s3n7, cis] = c_s3_input_size

c_array[c_s1n0, cos] = c_s1_output_size

c_array[c_s1n1, cos] = c_s1_output_size

c_array[c_s2n0, cos] = c_s2_output_size

c_array[c_s2n1, cos] = c_s2_output_size

c_array[c_s2n2, cos] = c_s2_output_size

c_array[c_s2n3, cos] = c_s2_output_size

c_array[c_s2n4, cos] = c_s2_output_size

c_array[c_s2n5, cos] = c_s2_output_size

c_array[c_s2n6, cos] = c_s2_output_size

c_array[c_s2n7, cos] = c_s2_output_size

bu_s1_array[1, ubn] = 25
bu_s1_array[1, lbn] = 33
bu_s1_array[2, ubn] = 26
bu_s1_array[2, lbn] = 34
bu_s1_array[3, ubn] = 27
bu_s1_array[3, lbn] = 35
bu_s1_array[4, ubn] = 28
bu_s1_array[4, lbn] = 36
bu_s1_array[5, ubn] = 29
bu_s1_array[5, lbn] = 37
bu_s1_array[6, ubn] = 30
bu_s1_array[6, lbn] = 38
bu_s1_array[7, ubn] = 31
bu_s1_array[7, lbn] = 39
bu_s1_array[8, ubn] = 32
bu_s1_array[8, lbn] = 40

bu_s2_array[1, ubn] = 41
bu_s2_array[1, lbn] = 45
bu_s2_array[2, ubn] = 42
bu_s2_array[2, lbn] = 46
bu_s2_array[3, ubn] = 43
bu_s2_array[3, lbn] = 47
bu_s2_array[4, ubn] = 44
bu_s2_array[4, lbn] = 48
bu_s2_array[5, ubn] = 49
bu_s2_array[5, lbn] = 53
bu_s2_array[6, ubn] = 50
bu_s2_array[6, lbn] = 54
bu_s2_array[7, ubn] = 51
bu_s2_array[7, lbn] = 55
bu_s2_array[8, ubn] = 52
bu_s2_array[8, lbn] = 56

bu_s2_array[1, scn] = c_s2n0
bu_s2_array[2, scn] = c_s2n0
bu_s2_array[3, scn] = c_s1n0
bu_s2_array[4, scn] = c_s1n0
bu_s2_array[5, scn] = c_s1n1
bu_s2_array[6, scn] = c_s1n1
bu_s2_array[7, scn] = c_s1n1
bu_s2_array[8, scn] = c_s1n1
bu_s2_array[1, con] = 0
bu_s2_array[2, con] = 1
bu_s2_array[3, con] = 2
bu_s2_array[4, con] = 3
bu_s2_array[5, con] = 0
bu_s2_array[6, con] = 1
bu_s2_array[7, con] = 2
bu_s2_array[8, con] = 3

bu_s3_array[1, ubn] = 57
bu_s3_array[1, lbn] = 59
bu_s3_array[2, ubn] = 58
bu_s3_array[2, lbn] = 60
bu_s3_array[3, ubn] = 61
bu_s3_array[3, lbn] = 63
bu_s3_array[4, ubn] = 62
bu_s3_array[4, lbn] = 64
bu_s3_array[5, ubn] = 65
bu_s3_array[5, lbn] = 67
bu_s3_array[6, ubn] = 66
bu_s3_array[6, lbn] = 68
bu_s3_array[7, ubn] = 69
bu_s3_array[7, lbn] = 71
bu_s3_array[8, ubn] = 70
bu_s3_array[8, lbn] = 72
bu_s3_array[1, scn] = c_s2n0
bu_s3_array[2, scn] = c_s2n0
bu_s3_array[3, scn] = c_s2n1
bu_s3_array[4, scn] = c_s2n1
bu_s3_array[5, scn] = c_s2n2
bu_s3_array[6, scn] = c_s2n2
bu_s3_array[7, scn] = c_s2n3
bu_s3_array[8, scn] = c_s2n3
bu_s3_array[1, con] = 0
bu_s3_array[2, con] = 1
bu_s3_array[3, con] = 0
bu_s3_array[4, con] = 1
bu_s3_array[5, con] = 0
bu_s3_array[6, con] = 1
bu_s3_array[7, con] = 0
bu_s3_array[8, con] = 1

i = 0
WHILE (i < 8) DO
{
  burst_array[i + 1, length] = 0
  burst_array[i + 1, separation] = 0
  burst_array[i + 1, destination] = 0
  INC i
c_s1_logic
#
# If a real cell was found, increment the real cells found counter and route the real cell
# else route the dummy cell to the dummy cell trap

if dummy = 0 THEN
   INC c_array[LOCATION()] - c_base_addr, rcf
    WAIT UNTIL c_cntrl_op = false
    ROUTE (c_array[LOCATION()] - c_base_addr, p2) + 1
    dummy = 1
    ROUTE (c_array[LOCATION()] - c_base_addr, p2) + 1
    c_array[LOCATION()] - c_base_addr, p2 = (c_array[LOCATION()] - c_base_addr, p2) + 1) MOD c_s1_output_size
ELSE
    ROUTE 5


c_s2_logic
#
# If a real cell was found, increment the real cells found counter and route the real cell
# else route the dummy cell to the dummy cell trap

if dummy = 0 THEN
    INC c_array[LOCATION()] - c_base_addr, rcf
    WAIT UNTIL c_cntrl_op = false
    ROUTE (c_array[LOCATION()] - c_base_addr, p2) + 1
    dummy = 1
    ROUTE (c_array[LOCATION()] - c_base_addr, p2) + 1
    c_array[LOCATION()] - c_base_addr, p2 = (c_array[LOCATION()] - c_base_addr, p2) + 1) MOD c_s2_output_size
ELSE
    ROUTE 3


c_s3_logic
#
# If a real cell was found, increment the real cells found counter and route the real cell
# else route the dummy cell to the dummy cell trap

if dummy = 0 THEN
    INC c_array[LOCATION()] - c_base_addr, rcf
    WAIT UNTIL c_cntrl_op = false
    ROUTE 1
)
ELSE
    ROUTE 2
c_cntrl_logic

# Declare local variables
INT _iba, _nin, _cis, _tmp

IF (ENTRIES() > stop_sim_count) THEN
  STOP

# Set the concentrator control operating flag to true and c_pointer to 1
  c_pointer = 1
  c_cntrl_op = true

WHILE (c_pointer <= no_concentrators) DO
  # Set the Operating flag and reset the Real Cells Found counter to 0, and initialize other
  # local concentrator parameters
  _iba = c_array[c_pointer, iba]
  _nin = c_array[c_pointer, nin]
  _cis = c_array[c_pointer, cis]
  
  IF (_hsa = false) THEN
  
  _hol = 1
  _hsa = true
  _p1 = (_p1 + 1) MOD _cis
  
  WAIT UNTIL ( _rca = true)
  _rca = false
  _rcf = 1
  
  ELSE
  
  _hol = 0
  _rcf = 0
  
  
  # If a request has been issued to reset the p2 pointer for the current concentrator, then
  # 1. Set the reset request flag to false
  # 2. Set the new value of p2 to that in p2rv, setting p2rv to 99 afterwards
  IF (_p2r = true) THEN
  
  _p2r = false
  _p2 = _p2rv
  _p2rv = 99
  
  
  # This loop examines all HOL cells until either the maximum number of real cells are
  # are found or all HOL cells have been exhausted, whatever comes first.
  DO
  
  _tmp = _iba + _p1
  
  IF (CONTENTS (LOC (_tmp)) > 0) THEN
  
  
  
  

SEND 1 ENT{ _p1 + 1} TO LOC(_nin)
WAIT UNTIL { _rca = true} OR { _hsa = false}

IF { _rca = true} THEN
{  
    _rca = false
    INC _hol
}
ELSE
    BREAK

} ELSE
    INC _hol

_p1 = { _p1 + 1) MOD _cis

UNTIL ( _hol = _cis)  

INC c_pointer
}
c_cntrl_op = false

c_pointer = 0

input_logic

IF { burst_array[_port, length] > 0 } THEN
{
    dummy = 0
    time_in = CLOCK()
    destination_addr = burst_array[_port, destination]
    DEC burst_array[_port, length]
}
ELSE
    burst_separation = P{mean_separation}
    WAIT burst_separation SEC
    burst_array[_port, length] = P{mean_length}
    burst_length = burst_array[_port, length]
    
    IF { burst_array[_port, length] > 0 } THEN
    {
        burst_array[_port, destination] = random_destination_addr(1)
        destination_addr = burst_array[_port, destination]
        DEC burst_array[_port, length]
        
        dummy = 0
        time_in = CLOCK()
    }
    ELSE
        dummy = 1
    }

ROUTE 1

IF { dummy = 0} THEN
    dummy = 1
ROUTE 1

output_logic

WAIT 1 SEC

# IF (destination_addr <> (LOCATION() - output_port_base_addr)) THEN
# DISPLAY "A cell destined for output port " $ destination_addr $ " arrived at port " $ (LOCATION()) - output_port_base_addr)

b_move_logic

cell_delay = CLOCK() - time_in

IF ( (dummy = 0) AND ( _rcf = _cos)) THEN
{ _hsa = false
  WAIT 1 SEC
}

bu_s1_logic

WAIT UNTIL (c_cntrl_op = false)

IF ( dummy = 0) THEN
{
  IF (bstst_s1) THEN
  { 
    IF ( CONTENTS ( _ubn_1 ) < b_s1_size ) THEN
      ROUTE 1
    ELSE
      ROUTE 3
  }
  ELSE
  { 
    IF ( CONTENTS ( _lbn_1 ) < b_s1_size ) THEN
      ROUTE 2
    ELSE
      ROUTE 3
  }
  }
ELSE
{ 
  IF (bstst_s1) THEN
  { 
    IF ( CONTENTS ( _lbn_1 ) < b_s1_size ) THEN
      ROUTE 2
    ELSE
      ROUTE 4
  }
  ELSE
  { 
    IF ( CONTENTS ( _ubn_1 ) < b_s1_size ) THEN
      ROUTE 1
    ELSE
      ROUTE 4
  }
  }
}

bu_s2_logic

WAIT UNTIL (c_cntrl_op = false)

IF ( dummy = 0) THEN
{
  IF (bstst_s2) THEN

IF ( CONTENTS ( _ubn_2 ) < b_s2_size ) THEN
  ROUTE 1
ELSE
  {  
    _set_p2r_s2
    _adjust_p2rv_s2
    ROUTE 3
  }
ELSE
  {  
    IF ( CONTENTS ( _lnb_2 ) < b_s2_size ) THEN
      ROUTE 2
    ELSE
      {  
        _set_p2r_s2
        _adjust_p2rv_s2
        ROUTE 3
      }
    }
  }
ELSE
  {  
    IF (btst_s2) THEN
      {  
        IF ( CONTENTS ( _lnb_2 ) < b_s2_size ) THEN
          ROUTE 2
        ELSE
          {  
            _set_p2r_s2
            _adjust_p2rv_s2
            ROUTE 4
          }
        }
      }
    ELSE
      {  
        IF ( CONTENTS ( _ubn_2 ) < b_s2_size ) THEN
          ROUTE 1
        ELSE
          {  
            _set_p2r_s2
            _adjust_p2rv_s2
            ROUTE 4
          }
        }
      }
    }
bu_s3_logic
  WAIT UNTIL {c_cntrl_op = false}
  IF (dummy = 0) THEN
  {  
    IF (btst_s3) THEN
      {  
        IF ( CONTENTS ( _ubn_3 ) < b_s3_size ) THEN
          ROUTE 1
        ELSE
          {  
            _set_p2r_s2
            _adjust_p2rv_s2
            ROUTE 4
          }
        }
      }
    }
  }
ELSE
{
    _set_p2r_s3
    _adjust_p2rv_s3
    ROUTE 3
}
ELSE
{
    IF ( CONTENTS ( _lbn_3 ) < b_s3_size ) THEN
        ROUTE 2
    ELSE
    {
        _set_p2r_s3
        _adjust_p2rv_s3
        ROUTE 3
    }
}
ELSE
{
    IF ( btst_s3 ) THEN
    {
        IF ( CONTENTS ( _lbn_3 ) < b_s3_size ) THEN
            ROUTE 2
        ELSE
        {
            _set_p2r_s3
            _adjust_p2rv_s3
            ROUTE 4
        }
    }
    ELSE
    {
        IF ( CONTENTS ( _ubn_3 ) < b_s3_size ) THEN
            ROUTE 1
        ELSE
        {
            _set_p2r_s3
            _adjust_p2rv_s3
            ROUTE 4
        }
    }
}
}

bu_s1_base_addr
bu_s2_base_addr
bu_s3_base_addr
c_base_addr
input_port_base_addr
output_port_base_addr
btst_s1
destination_addr < 4
btst_s2
(destination_addr MOD 4) < 2
btst_s3
(destination_addr MOD 2) = 0
no_concentrators 14
pl 1
p2 2
p2r 3
p2rv 4
hol 5
rcf 6
hsa 7
rca 8
iba 9
nin 10
cis 11
cos 12
ubn 1
lb 2
scn 3
con 4
length 1
separation 2
destination 3

_port
_pl
_p2
_p2r
_p2rv
_hol
_rcf
_hsa
_rca

 LOC (bu_s1_array[LOCATION()] - bu_s1_base_addr, ubn)

 LOC (bu_s2_array[LOCATION()] - bu_s2_base_addr, ubn)

 LOC (bu_s1_array[LOCATION()] - bu_s1_base_addr, ubn)

 LOC (bu_s1_array[LOCATION()] - bu_s1_base_addr, lb)

 LOC (bu_s2_array[LOCATION()] - bu_s2_base_addr, lb)

 LOC (bu_s3_array[LOCATION()] - bu_s3_base_addr, lb)

 c_array[bu_s2_array[LOCATION()] - bu_s2_base_addr, scn], p2r = 1
 c_array[bu_s3_array[LOCATION()] - bu_s3_base_addr, scn], p2r = 1

_adjust_p2r_s2

_IF (c_array[bu_s2_array[LOCATION()] - bu_s2_base_addr, scn], p2rv) > bu_s2_array[LOCATION()] - bu_s2_base_addr,

_adjust_p2rv_s2

_IF (c_array[bu_s3_array[LOCATION()] - bu_s3_base_addr, scn], p2rv) > bu_s3_array[LOCATION()] - bu_s3_base_addr,

_adjust_p2rv_s3

_IF (c_array[bu_s3_array[LOCATION()] - bu_s3_base_addr, scn], p2rv) > bu_s3_array[LOCATION()] - bu_s3_base_addr,

_adjust_p2r_s3

_IF (c_array[bu_s2_array[LOCATION()] - bu_s2_base_addr, scn], p2rv) > bu_s2_array[LOCATION()] - bu_s2_base_addr,
<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
<th>Cumulative</th>
<th>Percentage</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>random_destination_addr</td>
<td>Discrete</td>
<td>No</td>
<td>12.5</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>12.5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>12.5</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>12.5</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>12.5</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>12.5</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>12.5</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>12.5</td>
<td>7</td>
</tr>
</tbody>
</table>

Streams

<table>
<thead>
<tr>
<th>Stream #</th>
<th>Seed #</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>32</td>
<td>No</td>
</tr>
</tbody>
</table>
Appendix D  Special Thanks to

I would like to extend my sincere gratitude and thanks to the following people for their help and assistance in helping me complete this thesis.

My thesis committee for their assistance and guidance through this thesis...
Professor Roy Czernikowski
Professor George Brown
Professor Kevin Shank

ProModel and simulation consultant...
Professor Paul Stiebitz, Industrial Engineering Department

Statistical consultant...
Professor Joe Voelkel, Applied Statistics Department

Computer resources...
Vincent Incardona, Information Systems and Computing

My wife for her patience and unwavering support...
Michelle Krieger

My mother and father for supporting me through college...
Ken and Helen Krieger

My grandparents for their support and patience in waiting for my return...
William and Gladys Krieger
Winifred McCubbin

My Rochester family for their support through the last few months...
Dave and Anne Fabrizi