



NAVAL
POSTGRADUATE
SCHOOL

MONTEREY, CALIFORNIA

THESIS

DYNAMIC CHANNEL ALLOCATION

by

Andrew D. Kaminsky

September 2003

Thesis Advisor:
Second Reader:

John Gibson
Geoffrey Xie

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE September 2003	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE: Dynamic Channel Allocation			5. FUNDING NUMBERS
6. AUTHOR(S) Andrew D. Kaminsky			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			12b. DISTRIBUTION CODE
13. ABSTRACT (maximum 200 words) Dynamic Channel Allocation (DCA) offers the possibility of capturing unused channel capacity by allocating unused resources between competing network nodes. This can reduce or possibly eliminate channels sitting idle while information awaits transmission. This holds potential for increasing throughput on bandwidth constrained networks. The purpose of this thesis is to examine the techniques used to allocate channels on demand and access such methods ability to maximize throughput. This thesis will also explore potential benefits to be gained by DCA through the use of computer simulation.			
14. SUBJECT TERMS Dynamic Channel Allocation, Fixed Channel Allocation, Inverse Multiplexing			15. NUMBER OF PAGES 211
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

DYNAMIC CHANNEL ALLOCATION

Andrew D. Kaminsky
Captain, United States Army
B.S., Grand Valley State University, 1993

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2003

Author: Andrew D. Kaminsky

Approved by: John Gibson
Thesis Advisor

Geoffrey Xie
Second Reader

Peter Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Dynamic Channel Allocation (DCA) offers the possibility of capturing unused channel capacity by allocating unused resources between competing network nodes. This can reduce or possibly eliminate channels sitting idle while information awaits transmission. This holds potential for increasing throughput on bandwidth constrained networks.

The purpose of this thesis is to examine the techniques used to allocate channels on demand and assess such methods ability to maximize throughput. This thesis will also explore potential benefits to be gained by DCA through the use of computer simulation.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
	A. PREFACE.....	1
	B. PROBLEM STATEMENT	1
	C MOTIVATION	3
II.	BACKGROUND	5
	A. DEVELOPMENT	5
	B. BANDWIDTH.....	5
	C. UTILIZATION	6
	D. CHANNEL ALLOCATION	7
	1. Fixed Channel Allocation (FCA).....	7
	2. Dynamic Channel Allocation (DCA).....	8
	a. <i>Centralized Dynamic Channel Allocation</i>	9
	b. <i>Distributed Dynamic Channel Allocation</i>	11
	3. Comparison of FCA and DCA.....	11
	4. Hybrid Channel Allocation	12
	5. Algorithms	12
	E. TECHNIQUES.....	14
	1. Inverse Multiplexing at the Hardware Layer	14
	2. Inverse Multiplexing at the Data Link Layer.....	16
	a. <i>IMA</i>	16
	b. <i>Multilink PPP</i>	17
	c. <i>Multirate Service</i>	17
	3. Adaptive Inverse Multiplexing for Wide-Area Wireless Networks	18
	4. Local Multipoint Distribution Services.....	18
	5. Time Division Multiplexed on Demand	19
	6. Optically Interconnected Multiprocessors	19
	7. CSMA/CD-Based Multiple Network Lines	20
	8. Hybrid Channel Allocation in Wireless Networks.....	21
	9. Beowulf Ethernet Channel Bonding.....	22
	F. SUMMARY	23
III.	DETERMINING A SOLUTION	25
	A. HOW DOES A STATION RECEIVE A CHANNEL	25
	B. QUEUEING.....	25
	C. PROPAGATION DELAY.....	27
	D. SIMULATION DESIGN	27
	E. CLASS DESIGN	28
	1. Channel Allocation.....	30
	2. Display Channel	36
	3. Display Delivery Time	39

4.	First Come First Serve	41
5.	Fair Distribution	47
IV.	TESTING AND ANALYSIS OF RESULTS	53
A.	TRAFFIC GENERATION	53
B.	FIRST COME FIRST SERVE	55
1.	Testing	55
2.	Results	56
C.	FAIR DISTRIBUTION	65
1.	Testing	65
2.	Results	66
D.	ANALYSIS	78
E.	POSSIBLE IMPLEMENTATIONS	85
V.	CONCLUSION	87
A.	RECOMMENDATION	87
B.	FUTURE WORK	87
VI.	APPENDICES	89
A.	PROGRAM - JAVA CLASS: CHANNEL ALLOCATION	89
B.	PROGRAM – JAVA CLASS: DISPLAY CHANNEL	119
C.	PROGRAM – JAVA CLASS: DISPLAY DELIVERY TIME	137
D.	PROGRAM – JAVA CLASS: FIRST COME FIRST SERVE	162
E.	PROGRAM – JAVA CLASS: FAIR DISTRIBUTION	176
	LIST OF REFERENCES	191
	INITIAL DISTRIBUTION LIST	195

LIST OF FIGURES

Figure 1 Common channel utilization.....	2
Figure 2 Scenario for maximizing channel capacity.....	3
Figure 3 Time-slot assignment with reuse factor of 6	10
Figure 4 Generic diagram of using inverse multiplexers.....	14
Figure 5 Layer 1.5 of OSI model	19
Figure 6 Multiple Network Layout	20
Figure 7 Design of Program.....	28
Figure 8 Class Design	29
Figure 9 First-Come-First-Serve Flow Chart.....	42
Figure 10 First-Come-First Serve Pseudo-Code	44
Figure 11 Fair Distribution Flow Chart	48
Figure 12 Fair Distribution Pseudo-Code	50
Figure 13 Example of Data Traffic Generation on a Fixed Channel.....	54
Figure 14 Average Time Periods Taken for Delivery from Table 25 (light load).....	78
Figure 15 Decrease in Delivery Time from Table 25 (light load)	79
Figure 16 Average Time Periods Taken for Delivery from Table 26 (moderate load)	80
Figure 17 Decrease in Delivery Time from Table 26 (moderate load).....	81
Figure 18 Average Time Periods Taken for Delivery from Table 27 (heavy load)	82
Figure 19 Decrease in Delivery Time from Table 27 (heavy load).....	83

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1 Variables in Channel Allocation class	30
Table 2 Methods in Channel Allocation class	36
Table 3 Methods in Display Channel class.....	39
Table 4 Methods in Display Delivery Time class.....	41
Table 5 Methods in First Come First Serve class	47
Table 6 Methods in Fair Distribution class.....	52
Table 7 First Come First Serve Test Case A-H	55
Table 8 Test A with no dynamic channels.....	57
Table 9 Test B with 1 dynamic channel (16% of the total channels) using First-Come- First-Serve scheduling algorithm.....	58
Table 10 Test C with 2 dynamic channels (28% of the total channels) using First-Come- First-Serve scheduling algorithm.....	59
Table 11 Test D with 3 dynamic channels (38% of the total channels) using First-Come- First-Serve scheduling algorithm.....	60
Table 12 Test E with 4 dynamic channels (44% of the total channels) using First-Come- First-Serve scheduling algorithm.....	61
Table 13 Test F with 5 dynamic channels (50% of the total channels) using First-Come- First-Serve scheduling algorithm.....	62
Table 14 Test G with 10 dynamic channels (66% of the total channels) using First-Come- First-Serve scheduling algorithm.....	63
Table 15 Test H with 15 dynamic channels (75% of the total channels) using First-Come- First-Serve scheduling algorithm.....	64
Table 16 Fair Distribution Test Case A, J-N,P-Q.....	65
Table 17 Test A with no dynamic channels.....	67
Table 18 Test J with 1 dynamic channel (16% of the total channels) using Fair Distribution scheduling algorithm	68
Table 19 Test K with 2 dynamic channels (28% of the total channels) using Fair Distribution scheduling algorithm	69
Table 20 Test L with 3 dynamic channels (38% of the total channels) using Fair Distribution scheduling algorithm	70
Table 21 Test M with 4 dynamic channels (44% of the total channels) using Fair Distribution scheduling algorithm	71
Table 22 Test N with 5 dynamic channels (50% of the total channels) using Fair Distribution scheduling algorithm	72
Table 23 Test P with 10 dynamic channels (66% of the total channels) using Fair Distribution scheduling algorithm	73
Table 24 Test Q with 15 dynamic channels (75% of the total channels) using Fair Distribution scheduling algorithm	74
Table 25 Summary of 30 tests with a light traffic load.....	75
Table 26 Summary of 30 tests with a moderate traffic load	76
Table 27 Summary of 30 tests with a heavy traffic load	77

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ABBREVIATIONS

ARPANET	Advanced Research Projects Agency Network
ATM	Asynchronous Transfer Mode
BONDING	Bandwidth-On-Demand Interoperability Group
BRI	Basic Rate Interface
CCITT	Consultive Committee for International Telephone and Telegraph
CSMA/CD	Carrier Sense Multiple Access/Collision Detected
DAMA	Demand Assigned Multiple Access
DCA	Dynamic Channel Allocation
ESRA	Enhanced Staggered Resource Allocation
FCA	Fixed Channel Allocation
FDD	Frequency Division Duplexing
GUI	Graphical User Interface
HCA	Hybrid Channel Allocation
IEEE	Institute of Electrical and Electronics Engineers
IMA	Inverse Multiplexing over ATM
ISDN	Integrated Services Digital Network
KBPS	Kilobits per second
LMDS	Local Multipoint Distribution Services
MAC	Media Access Control
MBPS	Megabits per second
MLP	Multilink Point-to-Point Protocol
MP	Maximum Packing
MSC	Mobile Switch Center
OSI	Open Systems Interconnection
PRI	Primary Rate Interface
RFC	Request For Comments
RMON	Remote Monitoring
SAIN	Synchronous Adaptive Infrastructure Network

SNMP	Simple Network Management Protocol
SRA	Staggered Resource Allocation
TDD	Time Division Duplexing
TDM	Time Division Multiplexing
TDMA	Time Division Multiple Access
TSRP	Time Slot Reuse Partitioning
WDM	Wavelength Division Multiplexing
WWAN	Wide-area Wireless Access Networks

I. INTRODUCTION

A. PREFACE

Networking has been evolving over the past 30 years. One of the earlier protocols was the Aloha, which was introduced in 1970. Three years later, the Ethernet specification began and, although not the first effort in networking, it gained a dominant role in network field, gaining momentum over the Token Ring specification. The Open Systems Interconnection (OSI) Reference Model was defined in 1978 to provide a useful framework for visualizing the communications process and comparing products in terms of standards conformance and interoperability potential. This layered structure not only aids users in visualizing the communications process, it also provides vendors with the means for segmenting and allocating various communications requirements within a workable format [Ref. 1].

The efforts have produced a plethora of proposed standards, some which attained a dominant role in networking. However, it is usually the compromise of technology and business practices that leads to the emergence of standards. As a result, current standards are usually not the best means of accomplishing the task [Ref. 2]. An example of this is the IEEE 802.3 based on the Ethernet specification.

Today, the push in utilizing the potential of the Internet is conveying all types of traffic into a single networking fabric. Therefore, networks must increasingly deal with flows of real-time streaming audio, video, and multi-media traffic that requires low latency and an acceptable quality of service. Consequently, two primary considerations of networks are providing low latency connectivity and a dynamic bandwidth allocation to manage congestion where available bandwidth is limited [Ref. 3].

B. PROBLEM STATEMENT

The problem to be addressed by this thesis is whether or not holding some network capacity in reserve, such that it can be dynamically assigned to users based on offered load, can improve the delivery time of traffic in networks that experience significant propagation delays. This problem is particularly of interest in unguided media

such as wireless, underwater acoustics, satellite, and deep space communications where transmission channels have extreme propagation delays or interference with which to contend. Using several channels as a composite decreases the time needed for message delivery by decreasing the transmission delay.

Figure 1 illustrates the current problem. John and Carl are assigned Channel 1 and 2, respectively. Each is sending data of different sizes. John's data takes four time periods to send and Carl's data takes two time periods. While each person has his or her own dedicated channel, Channel 3 remains unused. Although this method accomplishes the task it is not the most efficient use of the bandwidth available, as some of the capacity remains idle even though there is traffic to be sent.

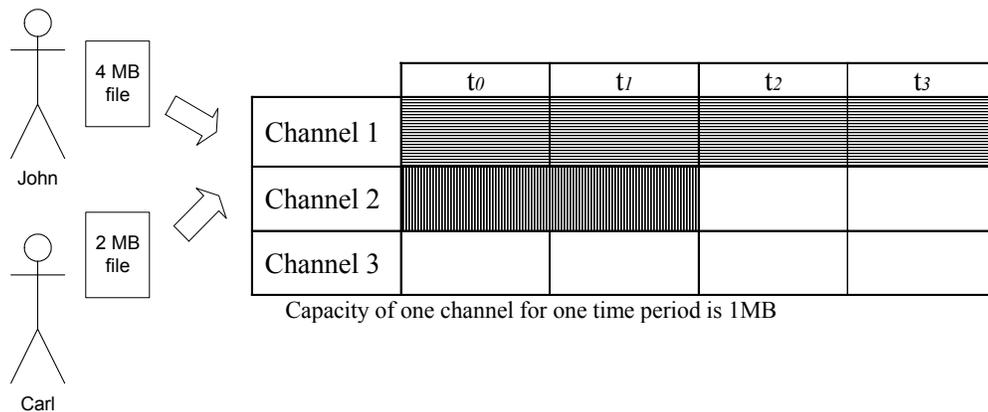


Figure 1 Common channel utilization

Another approach to this problem is to allocate channels efficiently based on demand. If the unused channel were dynamically allocated, then a more efficient scenario would be for Carl to use this extra capacity to send his entire data in one time period, t_0 . John would have to continue using his own dedicated channel until some unused channels become available. In time period, t_1 , two channels are unused and John can now send his remaining data. With this dynamic allocation of channels, better use of bandwidth is achieved, as shown in Figure 2.

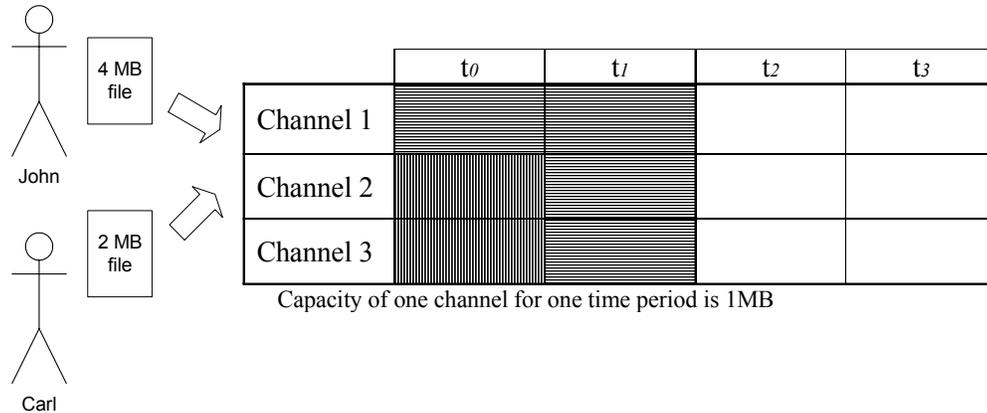


Figure 2 Scenario for maximizing channel capacity

However, in the event all the channels are assigned, if the user for channel 3 wants to send some data, then whoever is dynamically borrowing channel 3 must relinquish control of it and wait for it to become available again. Furthermore, if a channel is used for a session it cannot be used concurrently for other communications, as doing so would cause the data for each user of the shared channel to be corrupted.

C. MOTIVATION

The motivation for analyzing this problem is to increase the efficiency of the use of channel capacity. Given a limited amount of bandwidth, the need to use this resource in the most efficient means possible is important for maximizing the bandwidth utilization and minimizing delivery time.

This thesis begins by examining channel allocation techniques currently used. Following this, a solution is proposed that utilizes two types of algorithms to dynamically allocate channels from which a computer simulation is devised. Upon gathering the results, an analysis and recommendation are made and suggestions for future work in channel allocation.

Chapter II discusses and compares fixed channel allocation with dynamic channel allocation. In addition, some of the algorithms used to achieve channel allocation are explored. This chapter also looks at techniques used, such as inverse multiplexing, time division multiplexed on demand, hybrid channel allocation in wireless networks, and

Beowulf Ethernet channel bonding. The background information presented in this chapter gives an indication that channel allocation is still being examined and several implementations are proposed.

Chapter III proposes a solution with two scheduling algorithms. One algorithm is a First Come First Serve, which gives all free dynamic channels to the first requesting fixed channel. The other algorithm is a Fair Distribution, which gives a portion of the free dynamic channels to the requesting fixed channels. A computer program is devised that simulates these algorithms, and the methods used in the program classes are described in this chapter.

Chapter IV examines the results obtained from the computer simulation in Chapter III. A comparison is made between the two algorithms as well the ratio of fixed channels to dynamic channels. An analysis of the time taken between fixed and dynamic channels is also discussed.

The conclusion is presented in Chapter V. A recommendation is discussed as well as future work in dynamic channels. More research in areas of different algorithms, as well as data loss, channels of varying size, and quality of service may provide and improve performance with dynamic channel allocation.

II. BACKGROUND

A. DEVELOPMENT

In the late 1960s and early 1970s, a number of ideas and policies were developed for managing channels. K. Araki introduced the original Dynamic Channel Allocation (DCA) policy with wireless communications, which assigns to a new user any channel that is unused. D.C. Cox introduced the concepts of keeping channels in an order, assigning channels based on information about channel usage, and channel reassignment. J.S. Engel introduced the concept of initially assigning channels using Fixed Channel Allocation (FCA), but then allows a base station to borrow a channel from a neighboring cell if it has none available. Several other policies on managing channels have been developed over the last two decades, some requiring little to no information at one end of the admission control spectrum to the other end of spectrum in which complete knowledge of a network is required. The low end of the admission control spectrum is commonly referred with FCA, and moving towards the upper end places DCA with variations at the extreme end with Maximum Packing (MP) as the most complex. MP requires complete knowledge of all existing channel assignments in the entire system, and may potentially reassign all existing channels [Ref. 4].

B. BANDWIDTH

Bandwidth is a finite resource. It is a determining factor for the capacity of a data or voice channel. Ideally, if there were unlimited amounts of bandwidth available, performance of data communications would vastly improve. However, as this is not the case, maximizing the throughput over a finite amount of bandwidth is a problem of significant focus today. A key network performance parameter of interest to the end user is the delay experienced by traffic from its presentation by the source until its delivery at the intended destination.

When the Internet was created more than 25 years ago, it was called the ARPANET and was used almost exclusively by U.S. researchers and scholars for file transfer and E-mail. The bandwidth used was from 9.6 Kbps to 56 Kbps, which was

sufficient to support activities at that time. Today, individual and corporate users are flooding onto the Internet via the World Wide Web in increasing numbers. These users demand higher bandwidth-consuming technologies, such as multimedia applications [Ref. 5].

C. UTILIZATION

Efficient schemes must be developed in utilizing idle channels. Channel utilization may be considered as the total time spent sending original packets divided by the total time required to deliver the data. The higher the throughput for a given bandwidth, the higher the utilization achieved. Channel utilization is sometimes called network utilization. The network utilization can be monitored and measured by special equipment, such as protocol analyzers or remote monitoring (RMON) devices. In addition, certain hubs and switches provide network utilization statistics on their visual displays [Ref. 6].

Several communication protocols have been developed over the years, some emerging as adopted standards for communications. One of the earliest schemes developed was the Aloha protocol. The basic idea of the classic Aloha protocol is to transmit when desired, receive a positive acknowledgement from the receiver, and back off and retransmit if no acknowledgement is received (timeout). The channel utilization reaches its maximum value at .184, when the offered load reaches one frame per two frames periods for the network. A variation of the classic Aloha protocol is called Slotted Aloha. Slotted Aloha reduces the chance of collision and improves utilization up to a maximum value of .362. However, the mean delay is increased as a tradeoff [Ref. 7].

Another protocol is Demand Assigned Multiple Access (DAMA) in which the sender requests a reservation for a future time slot. When the time slot comes, the user transmits without contention. The tradeoff in this protocol is also higher latency. This technique is commonly used in satellite systems where there are more users than available channels [Ref. 7].

Carrier Sense Multiple Access/Collision Detection is a popular protocol in local area networking. This protocol requires the sender to listen to the carrier before

transmitting. When the channel is idle, the sender begins transmitting. If the sender detects a collision, the sender stops transmitting, waits a random period of time, and begins retransmitting. The collisions consume a very small percentage of available channel capacity, even under a moderate to heavy traffic load. Another way of looking at this is that collisions quickly redistribute the traffic load over the available time, maximizing channel utilization and application throughput [Ref. 6]. While this protocol works well in wired networks, there is a problem in wireless networks where the transmitter may not be able to detect a collision, depending upon the station's area of coverage.

Inhibit Sense/Multiple Access incorporates a base station that transmits a busy tone. With an absent busy tone, a user may transmit, however if a collision is detected the user backs off and retransmits. This is also known as Digital Sense Multiple Access [Ref. 7].

D. CHANNEL ALLOCATION

There are two major methods used to allocate channels: Fixed Channel Allocation (FCA) and Dynamic Channel Allocation (DCA). The two may be combined to yield a hybrid scheme. The purpose of these methods is to assign channels in such a way as to maximize capacity utilization, while at the same time maintaining communications quality. In most publications, channel allocation is applied to voice based systems where the aim is to minimize the call blocking and call-dropping rates. In contrast, the aim of channel allocation in a data oriented service is to improve the overall data throughput [Ref. 8].

1. Fixed Channel Allocation (FCA)

Fixed Channel Allocation's major advantage is simplicity. As its name implies, channels are fixed and dedicated to the specific user. This FCA policy clearly is sufficient to insure that no collisions occur on a given channel due to multiple users accessing it [Ref. 9]. One of its drawbacks, however, is that it is not adaptive to high traffic burstiness and thus has low efficiency when presented traffic loads, which do not present data according to constant bit, rate scales. Further, inefficiencies exist when

channel capacity is monolithic, while user's profiles vary with respect to presented loads. The lowered efficiency is exacerbated when the variance between user traffic loads is significant.

To overcome this low efficiency within the cellular telephony arena, there are two main schemes that can be applied to FCA: non-uniform channel allocation and channel borrowing. Non-uniform channel allocation takes into account the traffic distribution of cells. Cells with higher estimated traffic are assigned more channels, and each channel is dedicated to a source-destination pair. In the channel-borrowing scheme, when a cell has used up all its pre-assigned channels, it can satisfy additional demands by borrowing channels from a neighboring cell that has free channels. This enables the system to adapt to traffic demands. However, the stipulation in this is a channel can only be borrowed if the borrowed channel does not interfere with existing calls, including those in neighboring cells. Other neighboring cells are prohibited from using a channel that is borrowed, and therefore, the channel is known as a locked channel [Ref. 10]. Furthermore, the borrowed channels allow more users to be serviced by the cell, but does not necessarily address variance in individual user traffic presentations.

2. Dynamic Channel Allocation (DCA)

Dynamic Channel Allocation may help to overcome FCA's limited responsiveness or adaptability. In DCA, no channel is permanently assigned, as they are in FCA. The idea is based upon the establishment of a centrally managed channel pool for all users in a system. Channels are assigned as needed and returned back to the channel pool once they are no longer needed. The most common use of this is in cellular telephony, where a switch center allocates channels based on real-time channel requests from each cell.

In order to allocate a channel, information, either in the form of current measurements or from prior estimates, concerning the interference (congestion) or traffic data patterns is required. The more timely the information gathered, the better the channel allocation decision is likely to be [Ref. 8].

Two principal methods are used in DCA, centralized and distributed. The primary discriminator as to which method is employed is the type of information being supported, binary data or voice. Voice has traditionally utilized circuit switching, thus ensuring uniform delay throughout the session, whereas data utilizes packet switching where data, delays and packet loss rates may vary within a given communication session. However, even though voice requires a constant bit rate channel, it can be switched off during short pauses in speech, thereby creating bursts of approximately 10 to 20 ms of speech [Ref. 10].

a. Centralized Dynamic Channel Allocation

Centralized DCA utilizes a controller that assigns channels based upon specific costs. The cost function is different in different schemes. The centralized DCA schemes outlined by Katzela and Naghshineh assigns channels based on constraints such as reuse distance, future blocking probability, and the number of time of a channel is used. It can produce a near-optimum channel allocation. However, there is a potentially high cost associated with the overhead to accomplish this [Ref. 10].

One implementation of centralized assignment uses time slot reuse partitioning (TSRP). A time frame is divided into two portions, dedicated and shared. If a user requires a higher QoS, then that user would be given more timeslots in the shared portion of the frame. In a method called Staggered Resource Allocation (SRA), each cell is divided into six sections, although other divisors might be used. Timeslots are grouped into six sub-frames and sectors are labeled 1 to 6 counter-clockwise as shown in Figure 3.

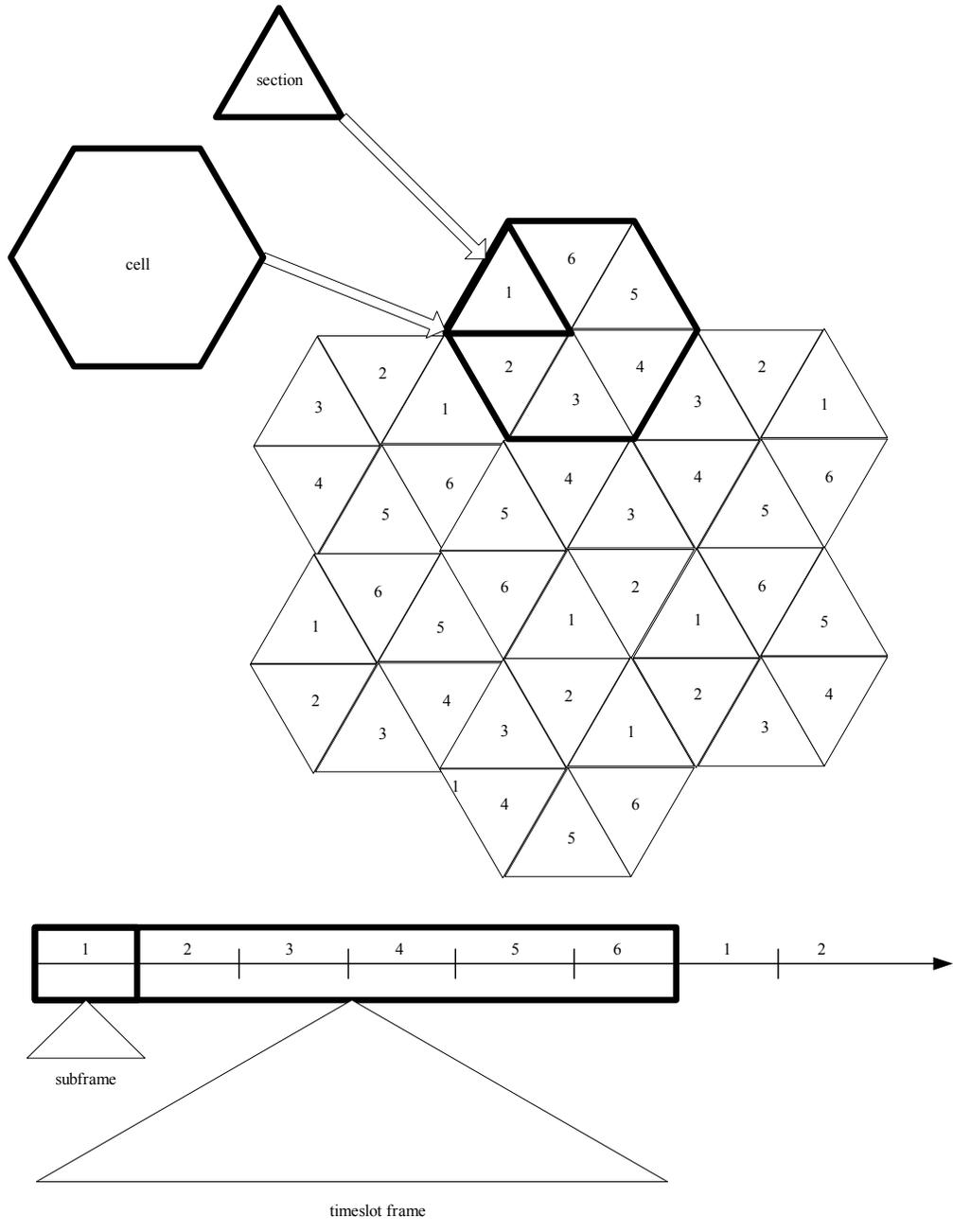


Figure 3 Time-slot assignment with reuse factor of 6

The sector labeling patterns for adjacent cells differ by a 120° rotation, thus creating a cluster of three cells whose patterns can be repeated across the entire system. Users in each sector can only transmit in the sub-frames dedicated to that sector. If they have additional data to send, it would be sent in another sub-frame in an order unique to each sector. The sub-frame is assigned to its sector in such a way as to minimize the interference between users. Each sector assigns time slots for transmitting packets to or from its terminals according to a special order. In a further refinement to this method, called Enhanced Staggered Resource Allocation (ESRA), the time sub-frames are further divided into mini-frames reducing concurrent transmission [Ref. 11].

b. Distributed Dynamic Channel Allocation

The distributed DCA scans for local information, such as signal strength, upon which to base channel allocations. Synchronization is critical to this technique. Ideally, scanning would occur before a transmission begins. However, with inference or bad feedback, efficiency of channel allocation would degrade. In addition, deadlocks and instability are more likely to occur in this method [Ref. 10].

3. Comparison of FCA and DCA

FCA is simpler than DCA to configure and implement. FCA also insures that no user can borrow or utilize a channel dedicated to another user, thus preventing that user from using its channel, which may happen under DCA.

DCA performs better than FCA under light or moderate loads, but FCA becomes superior under higher traffic loads. With heavy traffic load, FCA performs better when the maximum load is present in each cell, whereas, DCA usually does not conform to these constraints and so performs worse [Ref. 10]. Further, DCA methods require more overhead than FCA [Ref. 12]. Note that as the traffic load increases the traffic approaches more constant bit rate-like pattern.

The centralized DCA relies on a controller to manage the pool of channels, which can bring a network down if the controller fails. The distributed DCA relies on feedback, which can be distorted or delayed. Thus, the use of dynamic allocation does not come

without cost. It must be determined whether or not the cost exceeds the value of the benefit to be gained before implementing a dynamic allocation scheme.

4. Hybrid Channel Allocation

The Hybrid Channel Allocation (HCA) method makes use of the DCA method under light and medium traffic loads, and the FCA method under heavy traffic loading. Putting aside some channels for DCA and some for FCA can support this technique. The ratio of fixed-to-dynamic channels drives the performance of the system [Ref. 10].

5. Algorithms

The channel allocation problem is classified as an NP-complete (nondeterministic polynomial time) problem, which means as the size of the problem increases, the time required to solve the problem does not increase in a polynomial manner – but in an exponential manner. In fact, it is highly unlikely a polynomial time algorithm will be developed to exactly solve any NP-complete problem. However, if one or more input parameters can be adjusted, an acceptable or heuristic solution may be found. Such heuristic methods can solve a reasonable fraction of the common cases, where an approximation to the solution may be sufficient [Ref. 13].

In cellular systems, FCA is widely used because of its simplicity. However, it is not adaptive to time-dependent traffic. With DCA there is a centrally managed pool for all cells in a system and a Mobile Switch Center (MSC) that allocates channels based on real-time channel requests from each cell. The majority of algorithms in this area are for DCA, and they vary widely in their complexity. The simplest algorithms do not use any rearrangement when channels are allocated. They usually perform poorly in heavy traffic, even worse than FCA. The most complex algorithm of DCA uses “maximum packing” to allow every possible channel rearrangement to make room for new calls. As a result, channel capacity usage is at maximum. However, such algorithms are too complicated to implement [Ref. 14].

In a cellular phone example, a DCA algorithm is invoked and a series of channel readjustment are triggered for every event such as call origination, call ending, and handoff. To implement the readjustment, a lot of mobile subscribers and base stations are

involved in the channel handoff. These handoff activities not only degrade call quality, but also increase the loads to the mobile switch center and message networks. Due to these complexities, there is rarely a feasible DCA algorithm implemented in the real world. To allocate channels, DCA usually relies on the static and the partial interference information that was acquired before the system started. Therefore, the channel allocation in DCA is rarely optimized [Ref. 14].

The Genetic Algorithm is a heuristic-based algorithm. This algorithm is modeled after the natural process of evolution in which fitter individuals have a higher chance of reproducing and passing strong traits [Ref. 15]. It is able to find a good sub-optimal solution and is often referenced as a benchmark along with FCA for the other specific algorithms.

Another heuristic-based algorithm is the Tabu Search, which is a commonly used heuristic for solving combinatorial optimization problems. Its basic idea is to prevent cycling by forbidding or penalizing moves to previously visited points in the search space. In addition, unlike other search algorithms, where “best” generally means the best cost seen so far, Tabu Search selects “best” depending not only on cost evaluations but also on other conditions, such as the search history. The systematic use of memory is the major feature of Tabu Search. However, Tabu Search’s performance is satisfactory and not optimal [Ref. 15].

Some common algorithms for DCA are Random Assignment (RA) and Least Interference Algorithm (LIA). The RA is similar to the FCA Genetic Algorithm in that it does not require centralized knowledge of the network in order to allocate a channel, and as the name implies, a channel is randomly selected. RA is simple and effective for a system with a high number of available channels. The LIA, commonly employed in support of broadband fixed wireless access (BFWA), uses access points to measure the interference power of all available channels and selects the channel with the lowest interfered power [Ref. 8].

E. TECHNIQUES

An approach by Integrated Services Digital Network (ISDN) addresses the problem of bottlenecks or under utilized lines in the telephone network. Equipment that was non-ISDN and that did not operate at speeds of 56 or 64 Kbps under utilized the channel capacity. ISDN uses a technique called inverse multiplexing, otherwise known as bandwidth on demand [Ref. 16].

Inverse multiplexing speeds up data transmission by dividing a data stream into multiple concurrent streams that are transmitted at the same time across separate channels and are then reconstructed back into the original data stream at the other end as shown in Figure 4 [Ref. 17].

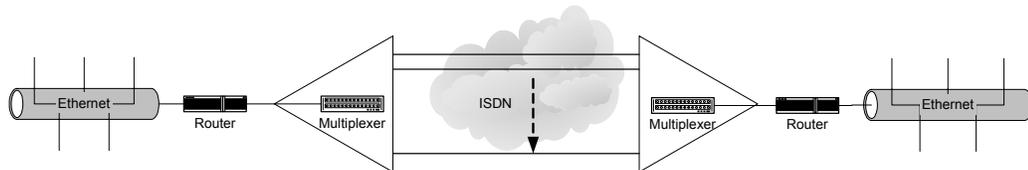


Figure 4 Generic diagram of using inverse multiplexers

1. Inverse Multiplexing at the Hardware Layer

ISDN is a set of digital transmission protocols defined by the Consultative Committee for International Telephone and Telegraph (CCITT has been renamed the Telecommunications Standards Sector of the International Telecommunications Union). It is the telephone network, turned digital from end-to-end, which transmits data and processes calls at significantly faster speeds and with greater clarity. ISDN utilizes the current twisted pair in many places today. However, specially designed equipment is necessary for the inverse multiplexing, currently available from vendors.

ISDN provides a raw data rate of 144 Kbps on a single pair twisted wire. Later it was discovered 160 Kbps could be squeezed out of the thin copper wire. This data rate is divided into two 56 Kbps or 64 Kbps channels (depending on the phone company), commonly called B, for bearer, channels, and a third channel of 16 Kbps, commonly called D, for delta, channel. The B channels carry voice or data and the D channel provides signaling and control. ISDN allows for dynamic bandwidth allocation to

increase effective throughput, which is the logical aggregation of both B channels. This type of interface is commonly called Basic Rate Interface (BRI) or Basic Rate Service (BRS).

The next higher capacity is Primary Rate Interface (PRI). In North America and Japan it is an aggregation of 23 B channels and 1 D channel (64 Kbps) yielding a total capacity of 1.544 Mbps, commonly called a T1 [Ref. 18].

The channel aggregation is also known as Bandwidth-On-Demand Interoperability Group (BONDING) or inverse multiplexing. It is the process of implementing a high speed channel by splitting it into several network channels at lower speeds and aggregating the network channels at the remote end. In 1991, a standard was defined for frame structure and procedures in establishing a wideband communications connection by combining multiple switched 56 and 64 Kbps channels through the use of an inverse multiplexer [Ref. 19]. In addition, the BONDING specifications implements four modes of operations for the inverse multiplexer:

- Mode 0 allows inverse multiplexers to receive two 56 Kbps calls from a video codec and initiate dual 56 Kbps calls to support a video conference.
- Mode 1 allows inverse multiplexers to spread a high speed data stream over multiple switched 56/64 Kbps circuits, but it does not provide error checking.
- Mode 2 adds error checking to each 56/64 Kbps circuit by withholding 1.6 percent of the bandwidth from each circuit for the passage of information that detects circuit failures and reestablishes links.
- Mode 3 uses out-of-band signaling for error checking, which may be derived from a separate dial-up circuit or the unused bandwidth of an existing circuit [Ref. 20].

An inverse multiplexer incorporates an administrative function, named a call profile. A call profile is a configurable file that contains the parameters of a particular data call so that a similar call can be established quickly. Also, remote administration can be performed on the inverse multiplexer, but at a small cost of the network's bandwidth. Most remote procedures can be performed using SNMP [Ref. 20].

There are many practical uses today for inverse multiplexing. For example, video conferencing and detailed imaging are becoming popular in which large amounts of data are needed. Another is dialed-up backup, where multiple lines are needed for large bandwidth in the event of a system crash [Ref. 21].

Some limitations do exist with inverse multiplexing. Channel characteristics may hinder performance of the overall transmission of a data group. Delays of an individual channel may cause delays in the demultiplexing and packet assembly. Also, inefficient scheduling algorithms may be constrained by the slowest link. Furthermore, loss of a channel may require retransmission of the entire data.

2. Inverse Multiplexing at the Data Link Layer

Inverse multiplexing originally began at the hardware layer, Layer 1 of the OSI model. This technique requires specific hardware technology be implemented. By implementing inverse multiplexing at the data link layer, Layer 2, software is used thereby making the development much easier. As a result, inverse multiplexing can be used over Asynchronous Transfer Mode (IMA), multilink PPP (MLP), or multilink frame relay (MFR). All three have some unique advantages, but each also carries some overhead [Ref. 22].

a. IMA

IMA specifies a transmission method in which ATM cells are fanned across several T1/E1 lines, and then reassembled at the receiving end without loss of the original ATM cell order. By enabling consolidated transport of the ATM protocol over T1 and E1 lines, IMA extends ATM to all portions of the WAN, not just to locations where traffic is very high or high capacity links are available.

Since the IMA access device at the receiving end requires a steady stream of cells to correctly recreate the original stream, the sending device introduces filler cells to keep the round-robin process at both ends in sync whenever there is a lull in traffic. To reduce bandwidth consumption, IMA removes idle and unassigned cells from the original stream and reinserts them at the receiving end.

Since IMA devices must operate across multiple networks, they must also operate under multiple network clocks, where non-synchronous circuits are routed through different paths and more than one timing domain. Each IMA receiving device must be able to implement controlled frame slippage to compensate for the timing differences between circuits and master clocks. They must also work in a hybrid network [Ref. 23].

Performance is one of the strongest points for IMA. Since it's based on ATM, all packets are 53 bytes long, so there's very low latency. In fact, IMA has all of the performance advantages of combining frame relay with fragmentation. However, IMA has the liability of ATM's relatively high overhead. This becomes significant when you have a high percentage of short packets that don't happen to fit nicely into ATM cell payloads [Ref. 22].

b. Multilink PPP

Multilink PPP (MLP) is simple and widely available, but the implementations are limited to IP traffic. Also, MLP tends to be implemented much more widely for lower-speed dial-up services than for higher-speed, dedicated access services. RFC 1717, written in 1994, defines MLP, addressing the aggregation of 64Kbps ISDN B channels into a logical, higher-bandwidth circuit. It was made obsolete by RFC 1990, which incorporated inverse multiplexing standards in 1996 [Ref. 24].

RFC 1618 recommends that the MLP be used instead of BONDING. This is because BONDING has an initialization period of its own, which might conflict with the simple detection technique of a configuration request being tried twice. In addition, BONDING requires extensive individual configuration in some current implementations when multiple B channels are used [Ref. 25].

c. Multirate Service

Multirate Service, sometimes called Nx64 service, is available from some telephone companies. With this service, a user receives a single channel, of whatever size, in multiples of 64 Kbps, on a per-call basis. This has the advantage that only one single call is made, resulting in a faster connection setup [Ref. 26]. However, this technique is both more expensive and less efficient than some of the others.

3. Adaptive Inverse Multiplexing for Wide-Area Wireless Networks

In wide-area wireless access networks (WWAN) inverse multiplexing is the standard method for providing higher end-to-end bandwidth. However, most WWAN use shared channels with highly variable link characteristics, including bandwidth, latency, and loss rates. An approach by Alex Snoeren, from the Massachusetts Institute of Technology, is to use a performance metric to adjust traffic scheduling, which he terms link quality balancing.

Snoeren's scheduling technique, similar to weighted round robin, is based on the ratio of the short-term average of observed throughput for each of the channels. Link quality balancing dynamically adjusts the MTU of each link in proportion to the available bandwidth. Splitting packets into fragments that can be transmitted and reassembled in roughly the same amount of time prevents slow channels from throttling the performance of the entire data stream [Ref. 27].

4. Local Multipoint Distribution Services

Local Multipoint Distribution Services (LMDS) is a relatively new type of terrestrial wireless service providing an attractive solution to the "last mile" problem of connecting consumers to broadband communications. In Leonidas Fountanas' thesis, *An Assessment of Emerging Wireless Broadband Technologies*, he describes LMDS as a full duplex system of the up and down streams of transmission and reception. Two common methods that he describes for this are Time Division Duplexing (TDD) and Frequency Division Duplexing (FDD). The TDD systems operate in a similar fashion to TDMA systems. The uplink and downlink use all the available bandwidth during transmission but only in specified time periods. Asymmetrical time slot allocation is possible for better utilization of the bandwidth, as uplinks typically require lower data rates. FDD systems provide two-way communication by separating the bandwidth into two smaller channels, one for reception and another for transmission. As in the TDD method, efficient use of the spectrum is feasible with asymmetrical bandwidth allocation for the two different channels. However, two separate antennas are generally required in FDD [Ref. 28].

5. Time Division Multiplexed on Demand

Time Division Multiplexed (TDM) Bandwidth on Demand is a technique developed to address broadband wireless access systems. It is a protocol that lies on top of the physical layer but below all higher level protocols, as shown in Figure 5.

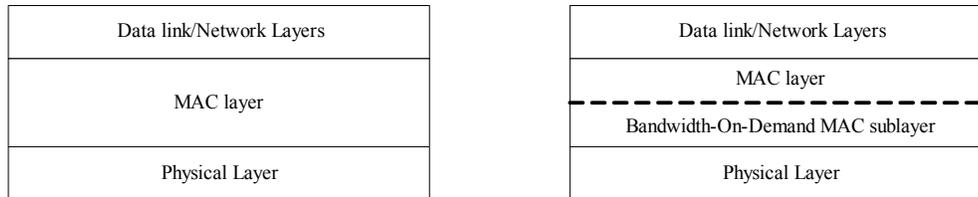


Figure 5 Layer 1.5 of OSI model

This approach assures non-obsolescence since it is completely independent from higher layer communications protocols. As a result, some refer to it as Layer 1.5 of the OSI model [Ref. 3]

Advantages of this approach include enhanced bandwidth utilization efficiency, low latency, and low delay variation. Broadband Wireless Access networks are a good application for this technique, as well as, Edge and Tandem node switches utilizing point to multipoint structures [Ref. 3].

The Synchronous Adaptive Infrastructure Network's (SAIN) approach uses the 1.5 Bandwidth-On-Demand sub-layer. It assigns an integer for cellets (TDM frames of small fixed length time slots) per frame for a single connection. As the value of the integer changes, the connection's bandwidth changes dynamically. SAIN proponents state the bandwidth management operation is very simple because it only needs a few bytes for most tasks [Ref. 29].

6. Optically Interconnected Multiprocessors

Optically interconnected networks, using wavelength division multiplexing (WDM), have potential connectability and reconfigurability capabilities that exceed electronic based networks. A study conducted by Joon-Ho Ha and Timothy Mark

Pinkston, from the University of Southern California, examined a token-based channel access protocol for WDM optically interconnected multiprocessors. Their study looked at the way TDMA passes control of channels between nodes with regard solely to time and not to communication needs. The token, as a supplemental control mechanism, passes control of channel access in conjunction with the slotted channel access of TDMA. This is referred to as token-based TDMA (T-TDMA). T-TDMA can accomplish higher channel utilization and lower latency, as the token enables the progressive recovery of unused slot spaces based on an individual node's communication needs. The use of a token in conjunction with TDMA has the potential to reduce packet latency by minimizing bandwidth waste due to unused slot spaces. A token is used to achieve dynamic allocation and recovery of unused slot space by allowing nodes without traffic to communicate to pass channel access onto other nodes needing to communicate. This technique makes it viable in the optical arena [Ref. 30].

7. CSMA/CD-Based Multiple Network Lines

Another approach in Ethernet networks is utilizing an alternative network topology devised by Gregory Cu and Nelson Marcos. This technique provides multiple network paths to all the stations connected to the network as shown in Figure 6.

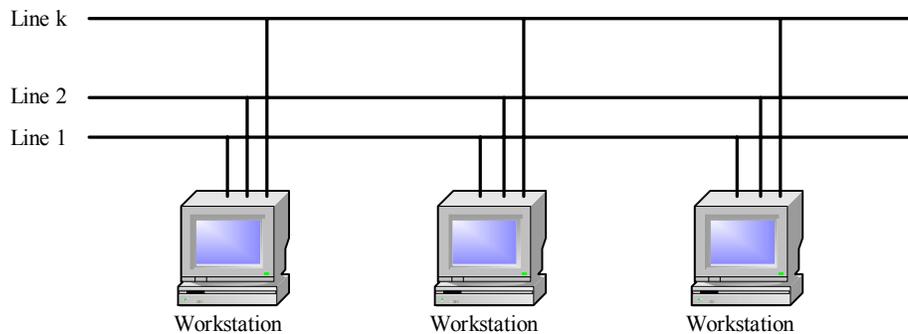


Figure 6 Multiple Network Layout

It is very much like a mesh network, however, in the physical layout, the workstations have several network devices, each attached to a single bus in a parallel bus

structure. The goal is to raise the bandwidth without using high-speed networks while utilizing current specifications and capabilities.

Cu and Marcos also developed a dynamic and scalable controller-algorithm for utilizing the CSMA/CD protocol on the networks. Their controller algorithm aimed at minimizing the transfer delay and collisions on the network. By minimizing transfer delay, the data to be transmitted would not stay in the computer's buffer too long, thereby increasing throughput. Minimizing collisions on the network lessens the retransmission of data by computers on the network. This increases throughput, thereby lessening the transfer delay.

The algorithm begins with no knowledge of the network. The first station to send data utilizes the first line available and upon completion of the data transfer gathers information about transfer delay and the number of collisions. This information is compared against set threshold values and stored for future reference. Then, over time, as the database statistics grows, a station will choose the best line available. There are four ways in which the station chooses:

- the highest most recent collision count and longest most recent transfer delay
- lowest most recent collision count and shortest most recent transfer delay
- highest average collision count and highest average transfer delay
- lowest average collision count and lowest average transfer delay

The performance increases as the number of network lines increases because stations distribute their data well among the network lines. However, since the existing CSMA/CD protocol is used, the more stations that utilize the medium, the more collisions occur, resulting in decreased throughput and transfer delay. Also based on a simulation analysis, smaller packet size had greater throughput [Ref. 31].

8. Hybrid Channel Allocation in Wireless Networks

The search for better channel allocation methods continues to be one of the major challenges in wireless communication network design. One such method is the hybrid channel allocation method, which uses co-channel distance as a criterion in both the

nominal channel allocation and the dynamic channel allocation in a local search algorithm. It tries to achieve high efficiency of channel use by assigning co-channels to cells that are close to each other. This method has two parts. The first part is the nominal channel allocation, in which a local search algorithm is employed. The second part deals with dynamic channel allocations, where minimizing co-channel distance is the goal.

The local search algorithm usually gives results with smaller co-channel distance. Therefore, channels are reused in cells that are closer to each other than in other nominal channel assignment schemes. As a result, a channel can be reused in more cells. The end result is more channels are available for dynamic allocation [Ref 32].

9. Beowulf Ethernet Channel Bonding

A by-product of using older computers in a NASA research lab resulted in a technique called Beowulf Ethernet Channel Bonding in 1994. This technique increases network bandwidth by utilizing clusters, which are any collection of more than one computer that can be accessed independently but also as a unit. In other words, it creates a single logical network by being transparent to the applications. Beowulf utilizes the Linux and Unix operating systems on an Ethernet-based network. To minimize protocol overhead and support the latest possible load-balancing decision making processes, Beowulf channel bonding is implemented at the device queue layer in the Linux kernel, below the IP protocol level. This has several advantages:

- Load-balancing may be done just before the hardware transmit queue, after the logical address (e.g. IP address) and the physical address (e.g. Ethernet station address) are added to the frame.
- Fragmented packets, usually large UDP/IP packets, take advantage of the multiple paths.
- Networks Stations that fail completely first fill their hardware queues, and are subsequently ignored [Ref. 33].

Channel bonding allows multiple network cards to be used as if they were one. Traffic is simply load-balanced over the multiple devices. There is no need to have one

separate switch per branch in the network. Further, for bonding to work, it is critical that all the boards in one machine have the same MAC (Ethernet) address. This would confuse a single switch, since it would not know to which port it should send a packet destined to a given MAC address [Ref. 34]. In other words, channel bonding combines multiple Ethernet connections between nodes into a single virtual channel similar to CSMA/CD based multiple network lines, in order to overcome bandwidth limitations, such as a 10Mbps Ethernet.

Linux is used since it proves to be robust, efficient, and ready to use. The availability of source code and limited licensing constraints permitted modification of the kernel to support multi-channel Ethernet communications. In a study of heterogeneous channel bonding on a Beowulf cluster, Baosong Zhao and Daniel Andersen, of Kansas State University, concluded that Gigabit Ethernet and channel bonding have greater network bandwidth than Fast Ethernet. Furthermore, they determined that multiple channels do scale in terms of both data bandwidth and transfer rate, and that packet size is very important for deriving the best performance from available communication resources [Ref. 35].

F. SUMMARY

This chapter examined channel allocation and techniques used. Two major methods of channel allocations are examined and compared: fixed and dynamic. Both methods have advantages and disadvantages, but the primary distinction was that DCA performs better under light or moderate loads where as FCA becomes superior under higher traffic loads.

The channel allocation problem is classified as an NP-complete problem. Therefore, an approximation to the solution may be sufficient by using heuristic methods.

Finally, this chapter examined some techniques currently employed. Inverse multiplexing, time division multiplexed on demand, CSMA/CD based multiple network lines, and hybrid channel allocation in wireless networks were some of the methods used in trying to maximize throughput.

The next chapter discusses one type of solution to channel allocation. Two types of scheduling algorithms are devised from which a computer simulation is developed. The algorithms are based on a first-come-first serve and a fair distribution from which a comparison is made between them as well as the number of dynamic channels available.

III. DETERMINING A SOLUTION

A. HOW DOES A STATION RECEIVE A CHANNEL

The first problem is to determine how a station will receive a channel. Our goal is to use dynamic channel allocation to recover some of the channel capacity wasted using full duplex. In addition, the synchronization must be looked at, using something such as a token. A control channel may also be introduced but this takes bandwidth and / or some associated overhead.

The two main approaches to wireless MAC protocols are mediated and contention. In a polling-based MAC protocol, one form of mediated access, a coordinator station is responsible for all the frame transmissions on the shared wireless medium. A wireless station that wants to transmit must wait until the coordinator station polls it. In contrast, a coordinator station is not required in a contention based protocol. Any wireless station that wishes to transmit does so if the wireless medium is available. The wireless stations are, in fact, contending for the shared medium, and thus, collisions are inevitable, as demonstrated by the Aloha protocol.

Fair allocation of bandwidth between channels and maximizing channel utilization are two important goals when designing a wireless MAC protocol. Unfortunately, there are inherent conflicts between these two design goals. For example, maximum channel utilization may be achieved if there is only one station transmitting continuously with zero back off, while all the other stations are starved. It is very difficult to maximize the channel utilization subject to the constraint of achieving fairness among traffic flows [Ref. 36].

B. QUEUEING

A queue is a storage of elements awaiting an action. Some common approaches to queuing are first come first serve, pushing and popping a stack, and priority. Tanenbaum characterizes queuing systems by five components:

- The interarrival-time probability density function
- The service-time probability density function
- The number of servers
- The queuing discipline
- The amount of buffer space in the queues [Ref. 37].

Queuing is typical in computer networking with packets. Most packet switches store-and-forward transmission at the inputs to the links. Store-and-forward transmission means that the switch must receive the entire packet before it can begin to transmit the first bit of the packet onto the outbound link. Thus, store-and-forward packet switches introduce a delay at the input to each link along the packet's route. This delay is proportional to the packet's length in bits. In addition, packets suffer output queuing delays, which are variable and depend on the level of congestion in the network [Ref. 38].

Tanenbaum mentions Kleinrock's queuing theory (1964) for a communication channel. He addresses the capacity of the channel and states that the mean packet size does not depend on the channel, as the capacity and input rate do. However, problems arise when several channels of varying size are encountered in a network.

Tanenbaum uses an example of dedicated versus shared channels. In his example, there are two computers connected by a 64 Kbps line. There are eight parallel sessions using the line. Each session generates Poisson traffic with a mean of 2 packets/sec. The packet lengths are exponentially distributed with a mean of 2000 bits. By using TDM or FDM, each 8 Kbps channel operates as an independent queueing system with $\lambda=2$ packets/sec and $\mu=4$ packet/sec, and the total time (queueing and transmission) is 500 msec. By using a single 64Kbps system, $\lambda=16$ packets/sec and $\mu=32$ packets/sec. resulting in a total time of 66.7 msec. By splitting up a single channel into k fixed size pieces makes the response time k times worse. The reason is that it frequently happens that several of the smaller channels are idle, while other ones are overloaded resulting in lost bandwidth that can never be regained [Ref. 37].

C. PROPAGATION DELAY

Propagation time is the measure of time required for a signal (or a bit) to travel from one point of the transmission medium to another. The propagation time is calculated by dividing the distance by the propagation speed. In addition, the processing time of a station, which normally is an insignificant value, contributes to the overall delay; however, this is not part of the propagation delay [Ref. 37].

The propagation delay is considered since it contributes to the total amount of time taken for the delivery of data. Different protocols and congestion flow may impact the number of propagation delays that must be considered, but the delay itself is a result of distance and speed only. For example, a protocol, such as Stop and Wait, that requires an acknowledgement for each packet is going to add significant delay since the amount of the propagation time doubles compared to a protocol that simply sends packets without acknowledgement or acknowledges at the end. Of course, the protocol that doesn't acknowledge each packet may require other actions to recover from packet loss.

D. SIMULATION DESIGN

The program developed is written in Java, SDK 1.4, from Sun Microsystems. The reason for this is the author is more familiar with Java than any other programming language. The design is based on a simple model shown in Figure 7 below:

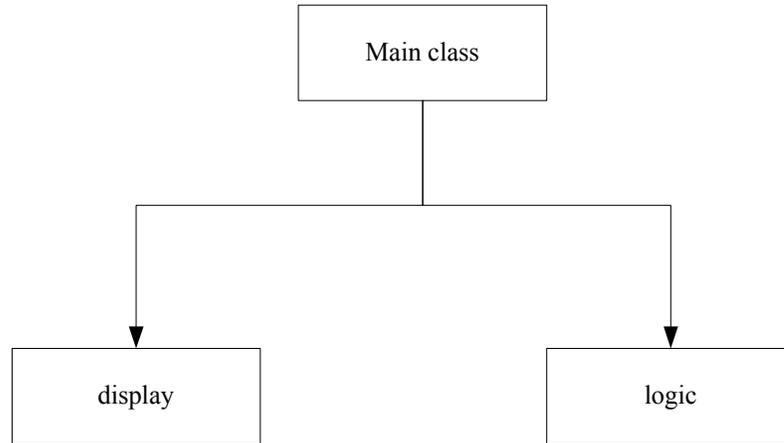


Figure 7 Design of Program

E. CLASS DESIGN

From this there are five classes created as shown in Figure 8. The main class is called ChannelAllocation. It is from this class the program begins execution. The display is shown through the DisplayChannel class that has a child class called DisplayDeliveryTime. These two classes display to the system screen a matrix of channels by time periods. In addition, a file is created identical to the screen display. The logic for manipulating the data elements (frames) by using dynamic channels is contained in the scheduling algorithms. In this case, there are two scheduling algorithms, one called FirstComeFirstServe and the other FairDistribution.

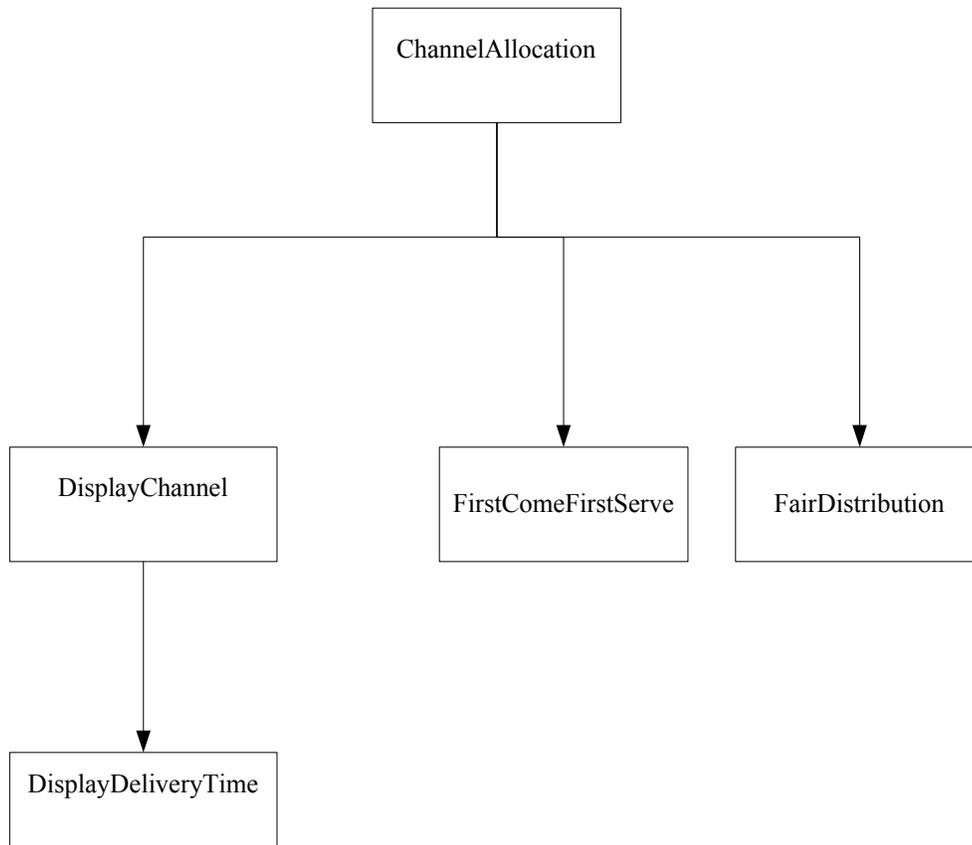


Figure 8 Class Design

The program centers on the matrix of channels and time periods. There are a finite number of channels and time periods. The simulation represents a bandwidth pipe shared between five hosts. The pipe is divided into channels, which are identified as fixed or dynamic. Each cell in the matrix can contain a data element (frame) and the block of data can span over several consecutive time periods. In addition, back-to-back data submission by individual hosts may occur. However, since this is a finite simulation, there are a fixed number of time periods.

Each time period reflects both the round-trip propagation delay between hosts and the transmission time of the data element. Transmission time of an acknowledgement is assumed to be negligible. Since an increase in the number of channels into which the pipe is divided proportionally decreases the transmission rate for the individual channels, the modeled time period must increase to reflect the change in transmission time. A

Stop-and-Wait protocol is assumed to simplify the allocation of data elements between the available channels.

Set values for the fixed channels are included in this design. One reason is for ensuring the program is working correctly, another is to set a reference. However, to make the simulation more realistic, data traffic randomness is used. This demonstrates dynamic channel allocation in progress.

The goal of creating this model is to see the effect of using dynamic channels and the impact, if any, of the time delivery. By adjusting the number of the dynamic channels an approximate ratio of fixed to dynamic channels may be obtained.

1. Channel Allocation

This is the main class from which the program executes. There are six static variables that are fixed and can be called from any of the other classes. Table 1 describes the variables.

Variables	
Name	Description
CHANNELS	The number of channels on the bandwidth
TIMEPERIODS	The number of time periods.
FIXCHANNELS	The number of fixed channels
SETTINGS	The identifying characteristics of the channel such as the type of channel (fixed or dynamic)
FIXCHANNELID	An abbreviation of “F” to denote fixed
DYNCHANNELID	An abbreviation of “D” to denote dynamic

Table 1 Variables in Channel Allocation class

Since this is the primary class the main method from which the program starts executing is included. There are administrative functions performed also, such as making sure a file can be written, assigning a string name for the file, and creation and initialization of arrays and strings.

A two dimensional array is created to represent channels and time periods. Since this is a simulation there are a finite number of time periods. This size of the matrix is determined by the static variables CHANNELS and TIMEPERIODS, defined in this class. This matrix is also a string type, since the values entered will be alphanumerical, a text prefix identifier, i.e., a or ab, and a sequence number. Therefore, a cell in the matrix may contain the values such as a0, a1, zB34. Another matrix is made identical to the main matrix, as a backup copy. Once the values in the main matrix are modified and another test needs to be run, the backup matrix can be used to copy the initial values back over to the main matrix. This is done to ensure the same sequence of data elements are used by each test run. The number of dynamic channels will be altered to provide an appropriate mix of fixed and dynamic channels. Note that the number of fixed channels remains constant across test runs to reflect the fact that each host has a dedicated channel assigned and then competes for access to the available dynamic channels.

Another two dimensional array is created to contain the characteristics of each channel. Therefore, the matrix is made by the static variable, CHANNELS x 2. This matrix holds a characteristic for each channel. This is the type, either fixed or dynamic. More characteristics can be implemented in the future by simply increasing the size of the matrix by CHANNELS x (n).

All the matrices are initialized with null values ensuring correct values held in memory. In addition, after each test is run, the matrices are reinitialized.

A string array is created to hold unique prefix identifiers. There are a set number defined in this array, but more can be added. When random values are generated a check is made to ensure an identical one isn't already being used. Alphabetical characters are used in combination with upper and lower case. For example, there are a, b, c, aa, bb, cc, Ab, Fta, ZZZ, etc.

Next there are four modal Graphical User Interfaces (GUIs) windows. When the program begins these windows appear asking the user (1) “Do you want random values generated”, (2) “ Do you want to append or overwrite an existing file, (3) “What type of scheduling algorithm do you wish to run, i.e., First Come First Serve, Fair Distribution, or both”, and (4) “the number of tests to run”. These allow the user to tailor the simulation run.

The class then runs a “baseline test” to show the progression of data transmission with fixed channels only. The backup matrix copy is also performed. Next, the simulation runs seven tests for each scheduling algorithm:

- one with 16% of the channels being allocated as dynamic (1 channel)
- one with 28% of the channels being allocated as dynamic (2 channels)
- one with 38% of the channels being allocated as dynamic (3 channels)
- one with 44% of the channels being allocated as dynamic (4 channels)
- one with 50% of the channels being allocated as dynamic (5 channels)
- one with 66% of the channels being allocated as dynamic (10 channels)
- one with 75% of the channels being allocated as dynamic (15 channels)

The key is that the number of fixed channels remains constant, and the number of dynamic channels determines the portion of the original capacity available to each channel.

Table 2 describes the methods in this class.

Methods		
Name	Return Type	Description
constructor		Default
start	void	This starts the program. It creates a two

Methods		
Name	Return Type	Description
		<p>dimensional array called a Channel Time Period matrix and initializes all the values to null. In addition, an identical array is made which will contain the same information as the original. This is copied back once the original has been modified. Another two dimensional array is created and called assignedChannel. It also has all its contents initialized to null. Another array is created to contain the Strings for prefix identifications.</p> <p>The user is prompted with four GUIs that ask if the user wants random values, to append a file, which scheduling algorithm to run, and the number of tests. Based on this the program runs and displays the contents before any scheduling algorithm takes place. Next the data in the Channel Time Period matrix is manipulated through 3 cases of the desired algorithm. One with 50% the channels assigned as dynamic, the second with 66% of the channel assigned as dynamic, and finally the third with 75% of the channels assigned as dynamic.</p>
userGeneratesRandomNumbers	boolean	This GUI asks the users whether or not to use random numbers to determine data arrival and duration values

Methods		
Name	Return Type	Description
userInputAppendFile	boolean	This GUI asks the user whether or not to append the results to an existing file
userInputSchedulingAlgorithm	int	This GUI asks the user to select a scheduling algorithm. This program provides two algorithms: First Come First Serve and Fair Distribution. In addition, both can be selected in a test run.
userInputNumberOfTests	int	This GUI asks the user to enter the number of tests to run.
ratioOfT	int	Calculates the ratio of fixed channels to total channels. For example, 5:5 is 1 (0% dynamic channels) 5:6 is 1.2 (16% dynamic channels) 5:7 is 1.4 (28% dynamic channels) 5:8 is 1.6 (38% dynamic channels) 5:9 is 1.8 (44% dynamic channels) 5:10 is 2 (50% dynamic channels) 5:15 is 3 (66% dynamic channels) 5:20 is 4 (75% dynamic channels)
changeNumberDynamicChannels	int	Changes the numbers of dynamic channels; currently, the program utilizes the numbers of 1, 2, 3, 4, 5, 10, and 15
createFileName	string	This creates a name for the output file

Methods		
Name	Return Type	Description
initializeOriginalTransferTime	void	Initializes the original transfer time matrix to zero values
initializeChannelTimePeriodMatrix	void	Initializes the Channel and Time Period matrix to null values
setSomeFixValuesChannelTimePeriodMatrix	void	Some defined values are set for representing traffic in the Channel and Time Period Matrix. This helps in troubleshooting and verifying the program
setSomeRandomValuesChannelTimePeriodMatrix	void	Random values are set for representing traffic in the Channel and Time Period Matrix
initializeAssignedChannel	void	Initializes the Assigned Channel matrix to null values
setAssignedChannel	void	Sets the assigned channel as fixed or dynamic
setInitialData	void	Sets the initial data in the Channel and Time Period matrix
copyOriginalMatrix	void	Copies the initial values placed in the Channel Time Period matrix to an identical array so the values can be used again in future scenarios
restoreOriginalMatrix	void	Restores the original values from the backup Channel Time Period matrix to the original Channel Time Period matrix

Methods		
Name	Return Type	Description
initializeMatrixForAnother Test	void	Initialize matrices for another test using the same original data
runBeforeTest	void	Runs the data before any scheduling algorithm begin. This provides a reference point against which to compare channel mixes
runTest	void	Runs the data through the scheduling algorithm

Table 2 Methods in Channel Allocation class

2. Display Channel

This class is responsible for displaying the contents of the main channel and time period matrix. It displays the output to a system screen, in the case of Windows it is the DOS window, and to a text file. Both outputs have the same information. For each test that is run a text file is created. For example, when the original test is run in which no dynamic channels are used, a text file is created. The seven tests with dynamic channels are appended after the original test. From the main channel allocation class, the user was prompted if he/she wants to append to an existing file. If the append is “YES” then when the next test is ran the output is appended to the end of the text file. In the case of “NO,” the current text file, if one exists, is overwritten when the test is ran.

There is a variable for displaying the screen width. In the original design, the width of the screen accommodated the size of the matrix. However, as the matrix expanded to as many as 50 time periods, it was not practical to display it all at one time. Rather, by changing the variable for displaying the screen width, the user can allow properly display to the screen, as well as the text file. For example, if the variable is set

to a value of 7, then the first 7 time periods are shown, then below that are the next 7, and below that the next 7, and so on until all time periods are displayed.

The following table describes the methods in this class.

Methods		
Name	Return Type	Description
constructor		Default
displayMatrix	void	Displays the main Channel and Time Period matrix both to the screen and a text file. This is the main function in this class that calls other functions within the class.
displayAssignedChannel	void	Displays the contents of the channel assignment (fixed or dynamic). This helps in troubleshooting.
calculateCurrentChannelUtilization	double	Calculates current channel utilization.
calculateCurrentTimePeriodUtilization	double	Calculates current time period utilization
calculateOverallUtilization	double	Calculates the overall utilization of the channels and time periods. This also helps in checking that the same value is returned in all the tests that are ran.
calculateDisplayBreaks	double	Calculates the total number of display breaks for the screen. This is useful when all the time periods in the Channel Time Period matrix cannot be properly displayed on a particular screen.

Methods		
Name	Return Type	Description
calculateBeginFrom	int	Calculates where the beginning of the matrix is for each display break. For example, if the desired display break is 7 then the beginFrom value will be 0 for the first display, 7 for the next, 14 for the next, and so on.
calculateEndAt	int	Calculates where the end of the matrix is for each display break. For example, if the desired display break is 7 then the endAt value will be 6 for the first display, 13 for the next, 20 for the next, and so on.
calculateLeadingSpacesINT	int	Calculates the number of the leading spaces to align the variable in the cell.
calculateTrailingSpacesINT	int	Calculates the number of the trailing spaces to align the variable in the cell.
calculateDigits	int	Calculates the number of digits in a number
findCellVariableSize	int	Calculates the number of characters in the cell.
calculateLeadingSpacesCHAR	int	Calculates the number of leading spaces to align the variable in the cell.
calculateTrailingSpacesCHAR	int	Calculates the number of trailing spaces to align the variable in the cell.
calculateNumberChannelsUsedPerTimePeriod	double	calculate the number of channels used during a time period

Methods		
Name	Return Type	Description
calculateNumberChannels Assigned	double	calculate the number of channels assigned either as fixed or dynamic
calculateBandwidthUtilization	double	calculate the bandwidth utilization

Table 3 Methods in Display Channel class

3. Display Delivery Time

This class is a child of the display channel class. It summarizes the values from the main channel and time period matrix. A table is created which contains (a) the data message prefix, (b) the size of the data message over time periods, (c) the size of the data message over time periods using fixed channels (d) the size of the data message over time periods using dynamic channels, (e) the RTT, and (f) the average of all the data messages.

The output is given after the main channel time period matrix is displayed. In addition to the screen display, an identical copy is appended to the text file.

Another text file is created with data generated from the tests. In the file are (a) the test number, (b) the scheduling algorithm used, (c) the number of fixed channels, (d) the number of dynamic channels, and (e) the average of all the data messages. Commas separate the five data elements. The file is designed to be imported into a Microsoft Excel spreadsheet.

Table 4 describes the methods in this class.

Methods		
Name	Return Type	Description

Methods		
Name	Return Type	Description
constructor		Default
delivery	void	Calculates and displays the delivery time through the use of dynamic channels.
dataFileName	string	Creates a name for a graph text file.
dataIdPrefix	string	Finds the data block identifiers (the prefix) in the Channel Time Period matrix
displayTimeDeliveryMatrix	void	Displays the contents of the delayDataID matrix and delayDataNumber matrix.
maxLength	int	Finds the maximum length of the variables stored in the array for proper display on the screen.
numberSpacesCHAR	int	Calculates the number of spaces for proper display on the screen using characters
numberSpacesINT	int	Calculates the number of spaces for proper display on the screen using integers
calculateLeadingSpaces	int	Calculates the number of leading spaces to align the variable in the cell
calculateTrailingSpaces	int	Calculates the number of trailing spaces to align the variable in the cell
storeDataIds	void	Store the data identifiers (prefixes) in the array.
countStoreDataBlockSize	void	Count the data block size for each data identifiers (prefixes) and store the value in the array

Methods		
Name	Return Type	Description
countStoreRTT	void	Count the time periods taken for the data block for each data identifiers (prefixes) and store the value in the array
calculateStoreT	void	calculate and store T
calculateDisplayThroughputChange	void	Calculate the delivery time change and display the result
calculateOverallDeliveryChange	double	Calculate the delivery time increase
calculateDeliveryTime	void	Calculate the delivery time: (RTT+T)
writeToExportDataFile	void	Export data to a file which will be used in a Microsoft Excel spreadsheet. The file has commas to delimitate between data
valueForGraph	void	Create (or append) a file for writing the results to. The results will be used for plotting a graph.

Table 4 Methods in Display Delivery Time class

4. First Come First Serve

The channel allocation class calls upon this class. The basic idea is the manipulation of the data in the fixed channels with the dynamic channels available. As the name of the class implies, it is a first come first serve scheduling algorithm. A flow chart of the algorithm is shown in Figure 9.

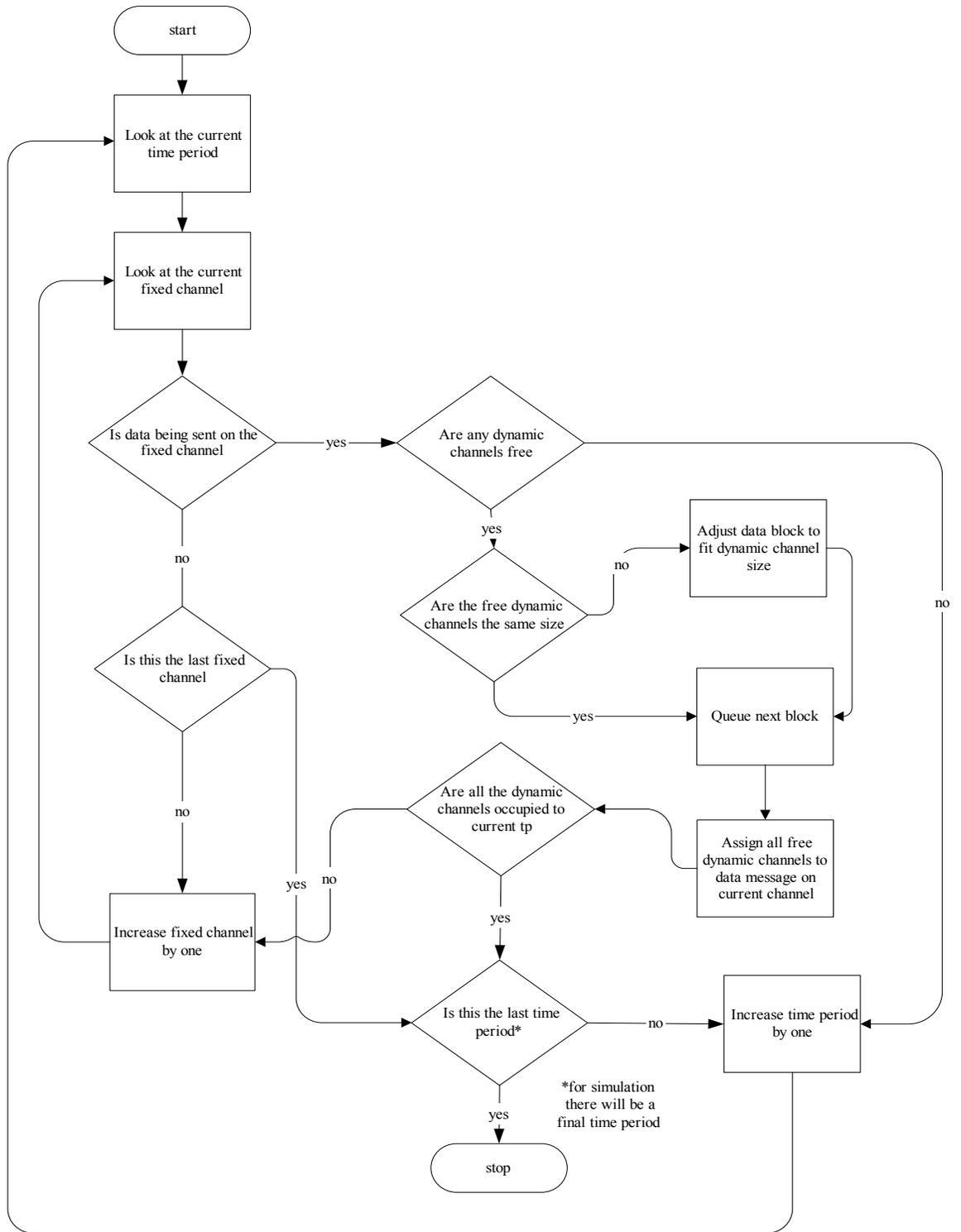


Figure 9 First-Come-First-Serve Flow Chart

The pseudo code for the algorithm is defined in Figure 10. It begins by looking at the first time period and then the first fixed channel in that time period. If no data is transmitted at this time, then the next fixed channel is examined, and the process continues until data is found. If no data is found on the fixed channels in the current time period, then the next time period is looked at in the same fashion. Once data is found on a fixed channel the data message is examined to see if it is longer than two time periods. The reason is if it were less than three time periods there would be insufficient time to allocate any dynamic channels. If the data message is longer than two time periods, the algorithm checks to see if any dynamic channels are free in the current time period. Once the conditions are satisfied, the elements of data message that are in future time periods are moved to the dynamic channels in the current time period. The algorithm allows the first data message to use as many dynamic channels as needed. If any dynamic channels are free in the current time period after the first message is done allocating channels, then the next data message is allowed to use the remaining free dynamic channels.

```

(begin at the first time period)
for i ← 1 to the number of time periods
  (begin at the first channel)
  for j ← 1 to the number of channels
    (found a fixed channel transmitting data)
    if the transmit message at i and j is NOT null AND
      the transmit message at i and j is a fixed channel
    then
      (if more than two data units of the message still needs to be transmitted
      that means the data  $\geq i+2$  can be sent dynamically)
      if more than two data units of the message still need to be
      transmitted
        (found free dynamic channels)
        if any dynamic channels are NULL
          (move the data message of  $i+2$  into dynamic
          channels)
          for k ← 1 to the number of free dynamic
          channels in i
            ► insert data units of  $i+2+k$  into
            dynamic channel
          (queue the remaining units of the data message into
          NULL time periods of the same channel)

```

```

if all the dynamic channels are NOT null
AND at least three more data units remain to
be transmitted
    for m ← to the number of remaining
    data units of the message
        ► insert data units of the
        message ≥ m into NULL time
        periods of the i+2 +m
    (increase the number of channel by 1)
    j = j+1
    (if at the end of the channels for the current time period, then increase the time period by
    1)
if at the end of the channels for current time period
then
    i = i + 1

```

Figure 10 First-Come-First Serve Pseudo-Code

From the algorithm the program class is created. The class examines the first fixed channel in the first time period. If data is found it is the first to use any dynamic channels. The data message size is determined. If the size is only two time periods long then no dynamic channels are given. This is to simulate that there is not enough time to allocate an additional channel, nor sufficient traffic to warrant its use if allocated. Therefore, any dynamic channels given a data message will be used to transmit data two frames after the current frame. That is, if a source has four frames waiting to be transmitted, only the third and fourth are eligible for transmission on dynamic channels.

With first come first serve, all the empty dynamic channels in $t_i + j$, where t_i is the current time period and j is the offset, and beyond are given to this data message. Once this is accomplished the algorithm examines the next channel in the current time period. If more than two time periods of data are found, then it will be allocated channels with the next free dynamic channels. No weight is given to one channel being more important than another channel, all are treated equally.

Table 5 describes the methods in this class.

Methods		
Name	Return Type	Description
constructor		Default
firstComeFirstServe	void	This is the method to run the First Come First Serve scheduling algorithm. It calls upon other methods within this class.
workDataMatrix	void	Copies the data block from the Channel Time Period matrix to a temporary Work Data matrix
moveWorkBackToFixedChannel	void	Copies the data block from the temporary Work Data matrix to the Channel Time Period matrix. It also indicates that particular a data block is completed by using “YES” or “NO”
displayWorkDataBlock	void	Displays the contents of the temporary Work Data matrix. This aids in troubleshooting
moveRemainingDataBlock	void	Copies the remaining data blocks from the temporary Work Data matrix back to the Channel Time Period matrix when there are no available dynamic channels. Thus, the data matrix serves as a queue for controlling data transmission
specialSwapDataCase	void	Ensure consecutive data elements are in correct order over the time periods
moveSameSizeEmptyDynamicChannel	void	Copies the data blocks from the temporary Work Data matrix back to the Channel

Methods		
Name	Return Type	Description
		Time Period matrix when there are available dynamic channels of the same size
checkCurrentChannelFixed	boolean	Checks if the current channel is fixed
checkCurrentChannelUsed	boolean	Checks if the current channel is used
countFutureData	int	Checks to see if the data message is longer than 2 time period, and if so may be used with free dynamic channels. The idea for this is since the current time period is happening at this time it is too late to use the empty channels, however, any channel in the future time periods may be used (perhaps, better termed: scheduled)
dataID	string	Determines the data block id used in the scenarios. For example, one data block sequence is z0, z1, z3 and another sequence is y1, y2, y3...
findSameSizeEmptyDynamicChannel	int	Finds an empty dynamic channel of the same size as data of a time period to be moved
anyNO	void	Finds any "NO" in the Work Data Block matrix. This may be due to not enough empty dynamic channels. Therefore, it places the data back into the main Channel Time Period matrix. As a result, as future time periods are examined this data may be

Methods		
Name	Return Type	Description
		reassigned if any future dynamic channels exist.

Table 5 Methods in First Come First Serve class

5. Fair Distribution

The channel allocation class calls upon this class. The basic idea is the manipulation of the data in the fixed channels with the dynamic channels available. As the name of the class implies, it is a fair distribution scheduling algorithm. This means the available dynamic channels are distributed fairly among fixed channels. A flow chart of the algorithm is shown in Figure 11

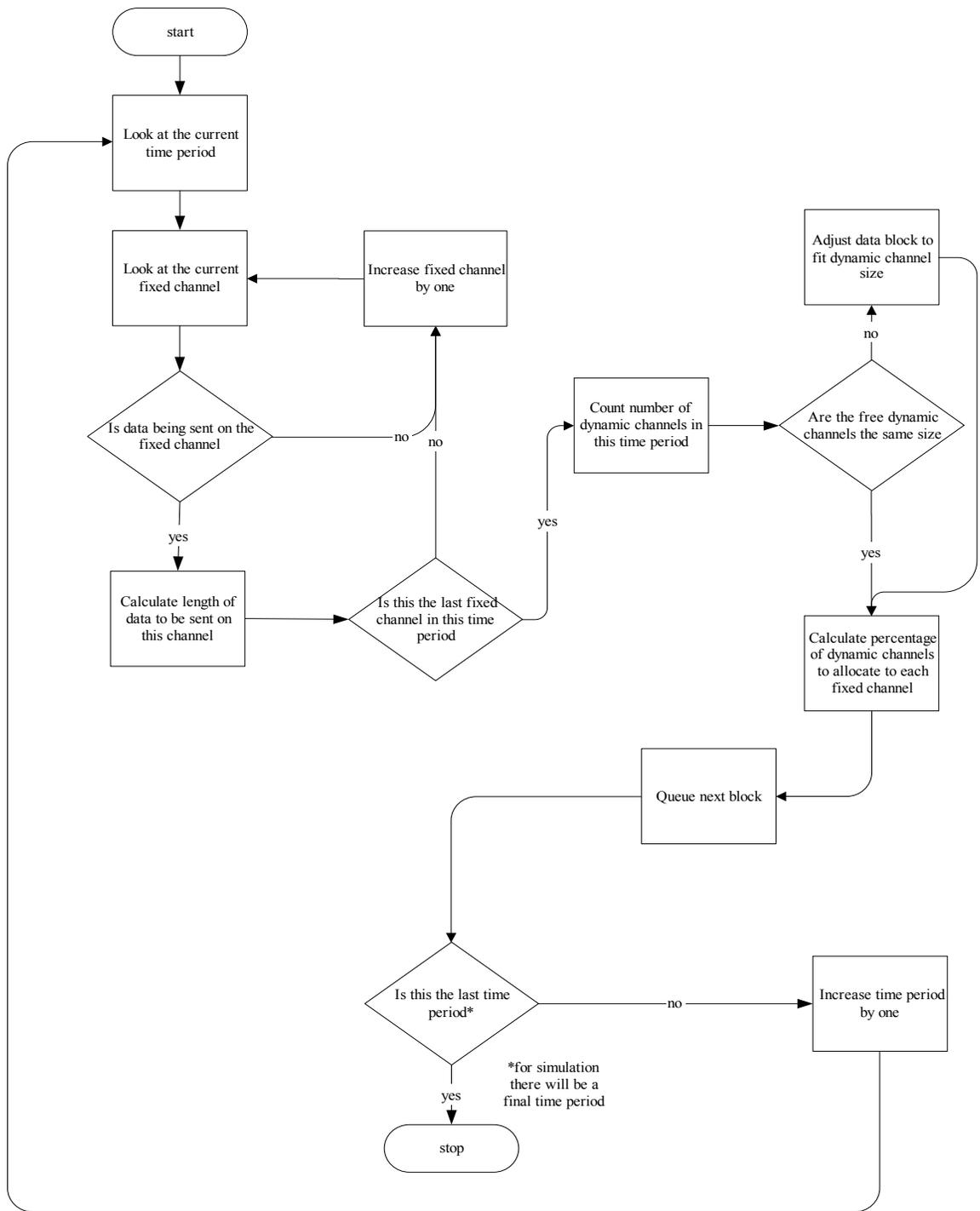


Figure 11 Fair Distribution Flow Chart

The pseudo code for the algorithm is defined in Figure 12. It begins by looking at the first time period and then the first fixed channel in that time period. If no data is transmitted at this time, then next fixed channel is examined, and the process continues until data is found. If no data is found on the fixed channels in the current time period, then the next time period is looked at in the same fashion. Once data is found on a fixed channel the data message is examined to see if it is longer than two time periods. The reason is if it were less than three time periods there would be insufficient time to allocate any dynamic channels. If the data message is longer than two time periods, then the size is temporarily stored. Once all the fixed channels have been examined in the current time period, a total for the size of all the messages is calculated. From this a portion of the dynamic channels is allocated to each fixed channel with a data message. As a result, a longer data message will get more dynamic channels than a shorter data message.

```

(begin at the first time period)
for i ← 1 to the number of time periods
  (begin at the first channel)
  for j ← 1 to the number of channels
    (found a fixed channel transmitting data)
    if the transmit message at i and j is NOT null AND
      the transmit message at i and j is a fixed channel
    then
      (if more than two data units of the message still needs to be transmitted
      that means the data ≥ i+2 can be sent dynamically)
      if more than two data units of the message still needs to be
        transmitted then temporarily store the size of the data
        message
      (increase the number of channel by 1)
      j = j+1
  (count the number of dynamic channels in this time period)
  for k ← to the number of dynamic channels
    (calculate the percentage of dynamic channels to allocate for each data block of
    the data message temporarily stored)
    for m ← to the number of data units of the data message
      temporarily stored
      (give a proportional fraction of the total available)
      allocate the number of dynamic channels based on the size
      of m by the total size of all the temporarily stored
      (move the data message blocks of i+2 into dynamic channels)
      ► insert data units of the message of m into dynamic channels
      allocated
  (queue remaining units of the data message into NULL time periods of the same channel)

```

```

if all the dynamic channels are NOT null AND
at least three more data units remain to be transmitted
    for n ←to the number of remaining units of the data message
        ►queue units of the data message ≥ n into NULL time
            periods of the i+2+n
        (increase the time period by 1)
    i = i + 1

```

Figure 12 Fair Distribution Pseudo-Code

The class begins by examining the first channel in the first time period. If data is found it examines the length of the data message. If it is less than three time periods (data units) long then no dynamic channels are allocated. It would only waste a dynamic channel. However, data of at least three units long is captured. Each channel is examined in the current time period. After this is accomplished a calculation is made on the number of dynamic channels to be allocated. If there are two data messages competing for dynamic channels the one with the most data to send will be given the most dynamic channels. For example, if there are two data messages, one of size 10 and one of size 5, then 2/3 of the dynamic channels are given to the one of size 10, and 1/3 are given to the one of size 5.

Table 6 describes the methods in this class.

Methods		
Name	Return Type	Description
constructor		Default
fairDistribution	void	This is the method to run the Fair Distribution scheduling algorithm. It calls upon other methods within this class.
countFixChannel	int	Counts the number of fixed channels in the assigned Channel matrix
countDynamicChannel	int	Counts the number of dynamic channels in

Methods		
Name	Return Type	Description
		the assigned Channel matrix
countFoundData	double	Counts the total number of future data block found during this particular time period
calculateDynamicChannel Allocation	void	Calculates the number of dynamic channels given to each channel needing future data to send.
displayFutureChannelMatrix	void	Displays the contents of the FutureChannel matrix. This aids in troubleshooting.
moveToDynamic	void	Moves a data block to a dynamic channel and pushes the remaining data block up one where there is a gap.
findDynamicChannel	int	Finds an empty dynamic channel in the current time period of the Channel Time Period matrix
moveFixedToDynamic	void	Moves the data element from the fixed channel to the dynamic channel
moveRemainingFixedUp	void	Moves the remaining data blocks up by one (to fill in the empty time period left by moving a data element to a dynamic channel – queue management)
checkCurrentChannelFixed	boolean	Checks if the current channel is fixed
checkCurrentChannelUsed	boolean	Checks if the current channel is used
countFutureData	int	Checks if at least three data units remain.

Methods		
Name	Return Type	Description
		The idea for this is since the current time period is happening at this time it is too late to use the empty channels, however, any channel in the future time periods may be used (perhaps, better termed: scheduled)
dataID	string	Determines the data block id sequence used in the scheduling algorithm. For example, one data block sequence is z0, z1, z3 and another is y1, y2, y3...

Table 6 Methods in Fair Distribution class

IV. TESTING AND ANALYSIS OF RESULTS

A. TRAFFIC GENERATION

Traffic generation is produced with randomness. To simulate varying degrees of traffic load, such as light, moderate, and heavy, a target utilization value is used. To simulate a light traffic load a low number such as 0.2 is used. A number such as 0.6 is used for simulating moderate traffic and 0.9 is used for heavy traffic.

The method begins to exam the amount of time periods available for a data message on a fixed channel. In the beginning it starts from the first time period and determines the number of time periods available. Next, a random message size is generated. The method has the option of setting a minimum data message size if required. The equation for this is:

$$\text{message size} = \text{uniform}(0, (\text{target utilization})(\text{time periods available}))$$

$$\text{message size} = \text{max}(\text{minimum message size}, \text{message size})$$

The final random step is to generate a start point. Of course, the start point must accommodate the data message size with the time periods available. For example, if only ten time periods are available and the data message size requires the occupation for eight of those time periods, then the start point cannot be later than time period 2. There is another option to limit the amount of space allowed to generate the start point called maximum message gap. The equation for this is:

$$\text{gap between messages} = \text{uniform}(0, \text{maximum message gap})$$

This can prevent a start point from occurring near the end of the space available, which may prevent more data messages from being produced. The next random data message begins after the previous data message.

Figure 12 shows an example of how random traffic may be generated on a fixed channel. In this example there are 500 time periods available. The first data message produced is “a”. A random size is generated for the data message, which in this case it occupies approximately $\frac{1}{4}$ of the space available. Next a random start point is generated,

which is shortly after the beginning time period. Again, since “a” has all the space available it may be better to use the option of setting the start point at an early point with the maximum message gap. Once data message “a” is completed, the method determines the free space available from the end of data message “a”. The same process continues and data message “b” is produced. Its start point begins after some empty time periods from the end of the data message “a”. Since there is still some available space after data message “b” another data message is produced, in this case “c”. The size of data message “c” occupies nearly all the space available. In addition, its random start point happens to begin immediately after data message “b”. Since there is very little time periods left after the end of data message “c” the method stops.

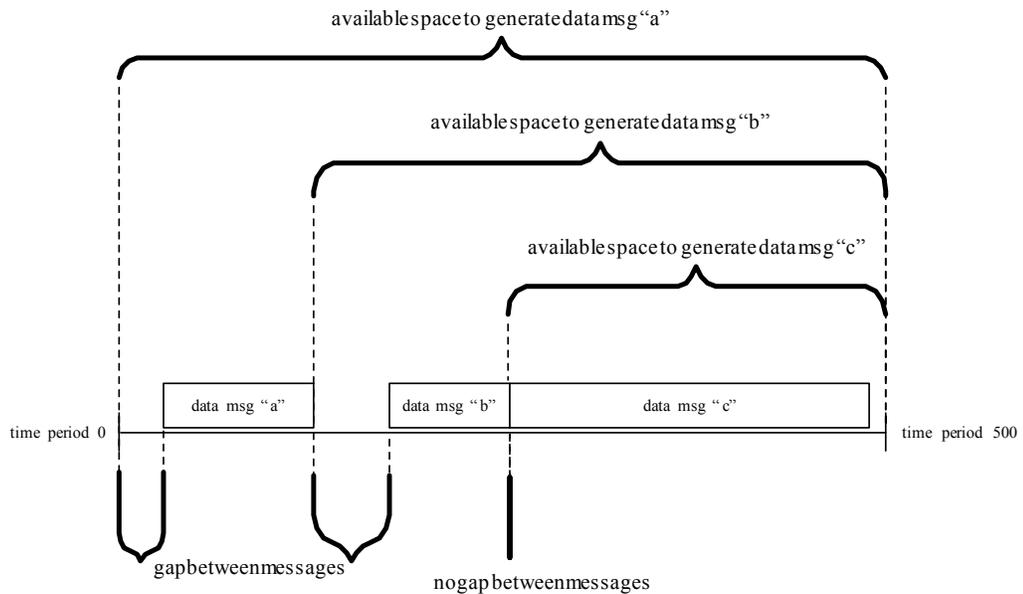


Figure 13 Example of Data Traffic Generation on a Fixed Channel

B FIRST COME FIRST SERVE

1. Testing

The first scheduling algorithm tested is the First-Come-First-Serve. The following eight tests are with this allocation method. The parameters used were:

Scheduling Algorithm:		First Come First Serve			
Test case identifier	Number of channels	Number of fixed channels	Number of dynamic channels	Percent of dynamic channels	Ratio of total to fixed channels
A	5	5	0	0%	1
B	6	5	1	16%	1.2
C	7	5	2	28%	1.4
D	8	5	3	38%	1.6
E	9	5	4	44%	1.8
F	10	5	5	50%	2
G	15	5	10	66%	3
H	20	5	15	75%	4

Table 7 First Come First Serve Test Case A-H

Test A is the base case from which the following seven tests are compared against. Test A does not use any dynamic channels. The following five tests increase the number of dynamic channels by one, and the next two increase the number of dynamic channels by five. Preliminary tests indicated the most change would likely occur when the number of dynamic channels was equal to or less than the number of fixed channels.

2. Results

The following tables represents the data traffic sent, as generated randomly for start time and duration. A random message size is generated to fit within the allotted time periods for each channel. The start time is randomly generated and no other random data message can begin until the a free time period exists, as well as ensuring there is no stepping over onto another data message. The channels are identified either as fixed (F) or dynamic (D). The size is relative to the transmission of a fixed size data unit (frame). Contiguous data units are identified by alphanumeric characters, such as b0, b1, b2 being a set of data and c0, c1, and c2 being another set of data. The percentage of channels used for a given time period, across all channels, is identified at the bottom of that time period's column. A summary is given at the bottom indicating the total number of data units in a given data message and the number of time periods taken to complete the data message delivery. In addition, the channels used are broken up into fixed and dynamic. As no specific transmission rate or propagation time is used, the actual expected times can be generated for a range of values. The very last item is the average of all the data messages over the delivery time.

Table 8 shows the output of Test A. Test A uses five fixed channels, which are denoted as “F” to the left of the channel in the matrix. There are no dynamic channels, and this test serves as the base case. There are seven data messages in this scenario, identified as “a”, “b”, “h”, “d”, “f”, “e”, and “i”. This test utilizes seven time periods, 0 to 6, for the sake of displaying the matrix on a page, however, more relevant data is obtained from increasing the time periods to a larger number. Increasing the time periods is done later in the test for the analysis. The summary table describes the data message size, the number of fixed channels used, the number of dynamic channels used, the RTT, and the average of all the data messages.

```
CA-bothScenarios.txt TEST #1
both scheduling algorithm
```

time period	0	1	2	3	4	5	6
F channel 1	a0	a1	a2	a3	a4	a5	a6
F channel 2		b0	b1	b2	b3	b4	
F channel 3	h0	h1	h2	h3		d0	d1
F channel 4	f0	f1	f2	e0	e1	e2	
F channel 5		i0	i1	i2	i3	i4	
# channels used	3	5	5	5	4	5	2
bw utilization	60%	100%	100%	100%	80%	100%	40%

data message prefix	a	b	h	d	f	e	i	avg
data message size	7	5	4	2	3	3	5	
data msg size (F)	7	5	4	2	3	3	5	
data msg size (D)	0	0	0	0	0	0	0	
RTT value: 1	7	5	4	2	3	3	5	4.143

Table 8 Test A with no dynamic channels

Table 9 represents Test B in which one dynamic channel (16% of the total channels) is used and the scheduling algorithm is the First-Come-First-Serve. Data message “a” is on the first channel which gets the first dynamic channel. No other data message is allowed to transmit on the dynamic channel as long as “a” has a transmit opportunity. It is not until time period 4 that data message “a” no longer needs a dynamic channel and the next data message is allowed to use the dynamic channel, in this case it is data message “b”. The average time has also decreased from the average time in Test A.

```
CA-bothScenarios.txt TEST #1
FirstComeFirstServe
```

time period	0	1	2	3	4	5	6	
F channel 1	a0	a1	a3	a5				
F channel 2		b0	b1	b2	b3			
F channel 3	h0	h1	h2	h3		d0	d1	
F channel 4	f0	f1	f2	e0	e1	e2		
F channel 5		i0	i1	i2	i3	i4		
D channel 6		a2	a4	a6	b4			
# channels used	3	6	6	6	4	3	1	
bw utilization	50%	100%	100%	100%	67%	50%	17%	
data message prefix	a	b	h	d	f	e	i	avg
data message size	7	5	4	2	3	3	5	
data msg size (F)	4	4	4	2	3	3	5	
data msg size (D)	3	1	0	0	0	0	0	
RTT value: 1	4	4	4	2	3	3	5	3.571

Table 9 Test B with 1 dynamic channel (16% of the total channels) using First-Come-First-Serve scheduling algorithm

Table 10 represents Test C in which two dynamic channels (28% of the total channels) are used and the scheduling algorithm is the First-Come-First-Serve. Again, data message “a” is on the first channel which gets the all the dynamic channels. No other data message is allowed to transmit on the dynamic channel as long as “a” has a transmit opportunity. In this case data message “a” finishes transmitting earlier than in Test B, and data message “b” begins using dynamic channels in time period 3 versus time period 4 in Test B. Again, the average time has decreased from the previous test.

```
CA-bothScenarios.txt TEST #1
FirstComeFirstServe
```

time period	0	1	2	3	4	5	6
F channel 1	a0	a1	a4				
F channel 2		b0	b1	b2			
F channel 3	h0	h1	h2	h3		d0	d1
F channel 4	f0	f1	f2	e0	e1		
F channel 5		i0	i1	i3	i2		
D channel 6		a2	a6	b4	i4		
D channel 7		a3	a5	b3	e2		
# channels used	3	7	7	6	4	1	1
bw utilization	43%	100%	100%	86%	57%	14%	14%

data message prefix	a	b	h	d	f	e	i	avg
data message size	7	5	4	2	3	3	5	
data msg size (F)	3	3	4	2	3	2	4	
data msg size (D)	4	2	0	0	0	1	1	
RTT value: 1	3	3	4	2	3	2	4	3.0

Table 10 Test C with 2 dynamic channels (28% of the total channels) using First-Come-First-Serve scheduling algorithm

Table 11 represents Test D in which three dynamic channels (38% of the total channels) are used and the scheduling algorithm is the First-Come-First-Serve. Again, data message “a” is on the first channel which gets the all the dynamic channels. No other data message is allowed to transmit on the dynamic channel as long as “a” has a transmit opportunity. In this case data message “a” finishes transmitting in time period 2, and data message “b” begins using dynamic channels in time period 2. Again, the average has decreased from the previous three tests.

CA-bothScenarios.txt TEST #1
FirstComeFirstServe

time period	0	1	2	3	4	5	6
F channel 1	a0	a1	a5				
F channel 2		b0	b1	b4			
F channel 3	h0	h1	h2	h3		d0	d1
F channel 4	f0	f1	f2	e0	e1		
F channel 5		i0	i1	i2			
D channel 6		a2	a6	i3	e2		
D channel 7		a3	b2	i4			
D channel 8		a4	b3				
# channels used	3	8	8	6	2	1	1
bw utilization	38%	100%	100%	75%	25%	13%	13%

data message prefix	a	b	h	d	f	e	i	avg
data message size	7	5	4	2	3	3	5	
data msg size (F)	3	3	4	2	3	2	3	
data msg size (D)	4	2	0	0	0	1	2	
RTT value: 1	3	3	4	2	3	2	3	2.857

Table 11 Test D with 3 dynamic channels (38% of the total channels) using First-Come-First-Serve scheduling algorithm

Table 12 represents Test E in which four dynamic channels (44% of the total channels) are used and the scheduling algorithm is the First-Come-First-Serve. Again, data message “a” is on the first channel which gets the all the dynamic channels. No other data message is allowed to transmit on the dynamic channel as long as “a” has a transmit opportunity. In this case data message “a” finishes transmitting on dynamic channels in time period 2, and data message “b” begins using dynamic channels in time period 2 and is able to complete transmission in time period 2. Another free dynamic channel exists in time period 2 and data message “h” utilizes it. Again, the average time has decreased from the previous four tests, but the change is not as much as previously observed.

```
CA-bothScenarios.txt TEST #1
FirstComeFirstServe
```

time period	0	1	2	3	4	5	6	
F channel 1	a0	a1	a6					
F channel 2		b0	b1					
F channel 3	h0	h1	h2			d0	d1	
F channel 4	f0	f1	f2	e0	e1			
F channel 5		i0	i1	i2				
D channel 6		a2	b2	i3	e2			
D channel 7		a3	b3	i4				
D channel 8		a4	b4					
D channel 9		a5	h3					
# channels used	3	9	9	4	2	1	1	
bw utilization	33%	100%	100%	44%	22%	11%	11%	
data message prefix	a	b	h	d	f	e	i	avg
data message size	7	5	4	2	3	3	5	
data msg size (F)	3	2	3	2	3	2	3	
data msg size (D)	4	3	1	0	0	1	2	
RTT value: 1	3	2	3	2	3	2	3	2.571

Table 12 Test E with 4 dynamic channels (44% of the total channels) using First-Come-First-Serve scheduling algorithm

Table 13 represents Test F in which five dynamic channels (50% of the total channels) are used and the scheduling algorithm is the First-Come-First-Serve. Again, data message “a” is on the first channel which gets the all the dynamic channels. No other data message is allowed to transmit on the dynamic channel as long as “a” has a transmit opportunity. In this case data message “a” finishes transmitting on the dynamic channels in time period 1, and data message “b” begins using dynamic channels in time period 2 and is able to complete transmission in time period 2. In addition, data message “h” uses a dynamic channel and completes its transmission, and data message “i” utilizes the last free dynamic channel in time period 2. Again, the average time has decreased from the previous five tests, but the change is not as much as previously observed.

```
CA-bothScenarios.txt TEST #1
FirstComeFirstServe
```

time period	0	1	2	3	4	5	6	
F channel 1	a0	a1						
F channel 2		b0	b1					
F channel 3	h0	h1	h2			d0	d1	
F channel 4	f0	f1	f2	e0	e1			
F channel 5		i0	i1	i3				
D channel 6		a2	b2	i4	e2			
D channel 7		a3	b3					
D channel 8		a4	b4					
D channel 9		a5	h3					
D channel 10		a6	i2					
# channels used	3	10	9	3	2	1	1	
bw utilization	30%	100%	90%	30%	20%	10%	10%	
data message prefix	a	b	h	d	f	e	i	avg
data message size	7	5	4	2	3	3	5	
data msg size (F)	2	2	3	2	3	2	3	
data msg size (D)	5	3	1	0	0	1	2	
RTT value: 1	2	2	3	2	3	2	3	2.429

Table 13 Test F with 5 dynamic channels (50% of the total channels) using First-Come-First-Serve scheduling algorithm

Table 14 represents Test G in which ten dynamic channels (66% of the total channels) are used and the scheduling algorithm is the First-Come-First-Serve. Again, data message “a” is on the first channel which gets the all the dynamic channels. No other data message is allowed to transmit on the dynamic channel as long as “a” has a transmit opportunity. In this case data message “a” finishes transmitting on dynamic channels in time period 1, and data messages “h” and “f” begins using dynamic channels in time period 1. Since no more transmit requirements exists in time period 1, two of the dynamic channels go unused. Again, the average time has decreased from the previous six tests, but the change is not as much as previously observed.

```
CA-bothScenarios.txt TEST #1
FirstComeFirstServe
```

time period	0	1	2	3	4	5	6	
F channel 1	a0	a1						
F channel 2		b0	b1					
F channel 3	h0	h1				d0	d1	
F channel 4	f0	f1		e0	e1			
F channel 5		i0	i1					
D channel 6		a2	b2		e2			
D channel 7		a3	b3					
D channel 8		a4	b4					
D channel 9		a5	i2					
D channel 10		a6	i3					
D channel 11		h2	i4					
D channel 12		h3						
D channel 13		f2						
D channel 14								
D channel 15								
# channels used	3	13	8	1	2	1	1	
bw utilization	20%	87%	53%	7%	13%	7%	7%	
data message prefix	a	b	h	d	f	e	i	avg
data message size	7	5	4	2	3	3	5	
data msg size (F)	2	2	2	2	2	2	2	
data msg size (D)	5	3	2	0	1	1	3	
RTT value: 1	2	2	2	2	2	2	2	2.0

Table 14 Test G with 10 dynamic channels (66% of the total channels) using First-Come-First-Serve scheduling algorithm

Table 15 represents Test H in which 15 dynamic channels (75% of the total channels) are used and the scheduling algorithm is the First-Come-First-Serve. Again, data message “a” is on the first channel which gets the all the dynamic channels. No other data message is allowed to transmit on the dynamic channel as long as “a” has a transmit opportunity. In this case data message “a” finishes transmitting on dynamic channels in time period 1, and data messages “h” and “f” begins using dynamic channels in time period 1. Since no more transmit opportunities exists in time period 1 six of the dynamic channels go unused. The average time did not change from the previous test.

```
CA-bothScenarios.txt TEST #1
FirstComeFirstServe
```

time period	0	1	2	3	4	5	6
F channel 1	a0	a1					
F channel 2		b0	b1				
F channel 3	h0	h1				d0	d1
F channel 4	f0	f1		e0	e1		
F channel 5		i0	i1				
D channel 6		a2	b2		e2		
D channel 7		a3	b3				
D channel 8		a4	b4				
D channel 9		a5	i2				
D channel 10		a6	i3				
D channel 11		h2	i4				
D channel 12		h3					
D channel 13		f2					
D channel 14							
D channel 15							
D channel 16							
D channel 17							
D channel 18							
D channel 19							
D channel 20							
# channels used	3	13	8	1	2	1	1
bw utilization	15%	65%	40%	5%	10%	5%	5%

data message prefix	a	b	h	d	f	e	i	avg
data message size	7	5	4	2	3	3	5	
data msg size (F)	2	2	2	2	2	2	2	
data msg size (D)	5	3	2	0	1	1	3	
RTT value: 1	2	2	2	2	2	2	2	2.0

Table 15 Test H with 15 dynamic channels (75% of the total channels) using First-Come-First-Serve scheduling algorithm

C. FAIR DISTRIBUTION

1. Testing

The next scheduling algorithm tested is the Fair Distribution. The following eight tests are with this allocation method as in the previous scheduling algorithm of First-Come-First-Serve. Since Test A uses no scheduling algorithm, the results are the same as shown in Table 8. The parameters used were:

Scheduling Algorithm:		Fair Distribution			
Test case identifier	Number of channels	Number of fixed channels	Number of dynamic channels	Percent of dynamic channels	Ratio of total to fixed channels
A	5	5	0	0%	1
J	6	5	1	16%	1.2
K	7	5	2	28%	1.4
L	8	5	3	38%	1.6
M	9	5	4	44%	1.8
N	10	5	5	50%	2
P	15	5	10	66%	3
Q	20	5	15	75%	4

Table 16 Fair Distribution Test Case A, J-N,P-Q

Test A is the base case from which the following seven tests are compared against. Test A does not use any dynamic channels. The following five tests increase the number of dynamic channels by one, and the next two increase the number of dynamic

channels by five. Preliminary tests indicated the most change would likely occur when the number of dynamic channels was equal to or less than the number of fixed channels.

2. Results

The following tables represent the data traffic sent, as generated randomly for start time and duration. A random message size is generated to fit within the allotted time periods for each channel. The start time is randomly generated and no other random data message can begin until a free time period exists, as well as ensuring there is no stepping over onto another data message. The channels are identified either as fixed (F) or dynamic (D). The size is relative to the transmission of a fixed size data unit (frame). Contiguous data units are identified by alphanumeric characters, such as b0, b1, b2 being a set of data and c0, c1, and c2 being another set of data. The percentage of channels used for a given time period, across all channels, is identified at the bottom of that time period's column. A summary is given at the bottom indicating the total number of data units in a given data message and the number of time periods taken to complete the data message delivery. In addition, the channels used are broken up into fixed and dynamic. As no specific transmission rate or propagation time is used, the actual expected times can be generated for a range of values. The very last item is the average of all the data messages over the delivery time.

Table 17 shows the output of Test A. Test A uses five fixed channels, which are denoted as “F” to the left of the channel in the matrix. There are no dynamic channels, and this test serves as the base case. There are seven data messages in this scenario, identified as “a”, “b”, “h”, “d”, “f”, “e”, and “i”. This test utilizes seven time periods, 0 to 6, for the sake of displaying the matrix on a page, however, more relevant data is obtained from increasing the time periods to a larger number. Increasing the time periods is done later in the test for the analysis. The summary table describes the data message size, the number of fixed channels used, the number of dynamic channels used, the RTT, and the average of all the data messages.

```

CA-bothScenarios.txt TEST #1
both scheduling algorithm

time period | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
-----|-----|-----|-----|-----|-----|-----|-----|
F channel 1 | a0 | a1 | a2 | a3 | a4 | a5 | a6 |
F channel 2 | | b0 | b1 | b2 | b3 | b4 | |
F channel 3 | h0 | h1 | h2 | h3 | | d0 | d1 |
F channel 4 | f0 | f1 | f2 | e0 | e1 | e2 | |
F channel 5 | | i0 | i1 | i2 | i3 | i4 | |
-----|-----|-----|-----|-----|-----|-----|
# channels used | 3 | 5 | 5 | 5 | 4 | 5 | 2 |
bw utilization | 60% | 100% | 100% | 100% | 80% | 100% | 40% |

data message prefix | a | b | h | d | f | e | i | avg
-----|-----|-----|-----|-----|-----|-----|-----|
data message size | 7 | 5 | 4 | 2 | 3 | 3 | 5 |
data msg size (F) | 7 | 5 | 4 | 2 | 3 | 3 | 5 |
data msg size (D) | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
RTT value: 1 | 7 | 5 | 4 | 2 | 3 | 3 | 5 | 4.143
-----|-----|-----|-----|-----|-----|-----|

```

Table 17 Test A with no dynamic channels

Table 18 represents Test J in which one dynamic channel (16% of the total channels) is used and the scheduling algorithm is the Fair Distribution. Data message “a” is the largest data message with a transmit opportunity in time period 0. Therefore, data message “a” gets the first dynamic channel. Data message “a” continues to dominant the dynamic channel as it is the largest data message. If two data messages with a transmit opportunity exists, then the free dynamic channel goes to the first channel checked. In this case the check is in numerical order. It is not until time period 4 that data message “a” no longer needs a dynamic channel and the next largest data message is allowed to use the dynamic channel, in this case it is data message “b”. The average time has also decreased from the average time in Test A.

```
CA-bothScenarios.txt TEST #1
FairDistribution
```

time period	0	1	2	3	4	5	6	
F channel 1	a0	a1	a3	a5				
F channel 2		b0	b1	b2	b3			
F channel 3	h0	h1	h2	h3		d0	d1	
F channel 4	f0	f1	f2	e0	e1	e2		
F channel 5		i0	i1	i2	i3	i4		
D channel 6		a2	a4	a6	b4			
# channels used	3	6	6	6	4	3	1	
bw utilization	50%	100%	100%	100%	67%	50%	17%	
data message prefix	a	b	h	d	f	e	i	avg
data message size	7	5	4	2	3	3	5	
data msg size (F)	4	4	4	2	3	3	5	
data msg size (D)	3	1	0	0	0	0	0	
RTT value: 1	4	4	4	2	3	3	5	3.571

Table 18 Test J with 1 dynamic channel (16% of the total channels) using Fair Distribution scheduling algorithm

Table 19 represents Test K in which two dynamic channels (28% of the total channels) are used and the scheduling algorithm is the Fair Distribution. Since there are two dynamic channels and three data messages competing for them in time period 0, a portion is given to only two data message as in the case of data messages “a” and “h” in time period 1. Again, the average time has decreased from the previous test.

CA-bothScenarios.txt TEST #1
FairDistribution

time period	0	1	2	3	4	5	6
F channel 1	a0	a1	a3	a5			
F channel 2		b0	b1	b3			
F channel 3	h0	h1	h3			d0	d1
F channel 4	f0	f1	f2	e0	e1		
F channel 5		i0	i1	i2	i3		
D channel 6		a2	a4	a6	e2		
D channel 7		h2	b2	b4	i4		
# channels used	3	7	7	6	4	1	1
bw utilization	43%	100%	100%	86%	57%	14%	14%

data message prefix	a	b	h	d	f	e	i	avg
data message size	7	5	4	2	3	3	5	
data msg size (F)	4	3	3	2	3	2	4	
data msg size (D)	3	2	1	0	0	1	1	
RTT value: 1	4	3	3	2	3	2	4	3.0

Table 19 Test K with 2 dynamic channels (28% of the total channels) using Fair Distribution scheduling algorithm

Table 20 represents Test L in which three dynamic channels (38% of the total channels) are used and the scheduling algorithm is the Fair Distribution. Now there are three dynamic channels and three data messages competing for them in time period 0. However, since data message “a” is larger than data messages “h” and “f”, a larger portion is given to “a”, and “h” gets the remaining one dynamic channel in time period 1. Again, the average time has decreased from the previous test.

```
CA-bothScenarios.txt TEST #1
FairDistribution
```

time period	0	1	2	3	4	5	6
F channel 1	a0	a1	a4	a6			
F channel 2		b0	b1	b3			
F channel 3	h0	h1	h3			d0	d1
F channel 4	f0	f1	f2	e0	e1		
F channel 5		i0	i1	i3			
D channel 6		a2	a5	b4	e2		
D channel 7		a3	b2	i4			
D channel 8		h2	i2				
# channels used	3	8	8	6	2	1	1
bw utilization	38%	100%	100%	75%	25%	13%	13%

data message prefix	a	b	h	d	f	e	i	avg
data message size	7	5	4	2	3	3	5	
data msg size (F)	4	3	3	2	3	2	3	
data msg size (D)	3	2	1	0	0	1	2	
RTT value: 1	4	3	3	2	3	2	3	2.857

Table 20 Test L with 3 dynamic channels (38% of the total channels) using Fair Distribution scheduling algorithm

Table 21 represents Test M in which four dynamic channels (44% of the total channels) are used and the scheduling algorithm is the Fair Distribution. With four dynamic channels, data message “a” still gets the larger portion of the dynamic channels in time period 1. Again, the average time has decreased from the previous test.

```
CA-bothScenarios.txt TEST #1
FairDistribution
```

time period	0	1	2	3	4	5	6
F channel 1	a0	a1	a5				
F channel 2		b0	b1	b4			
F channel 3	h0	h1	h3			d0	d1
F channel 4	f0	f1	f2	e0	e1		
F channel 5		i0	i1	i3			
D channel 6		a2	a6	i4	e2		
D channel 7		a3	b2				
D channel 8		a4	b3				
D channel 9		h2	i2				
# channels used	3	9	9	4	2	1	1
bw utilization	33%	100%	100%	44%	22%	11%	11%

data message prefix	a	b	h	d	f	e	i	avg
data message size	7	5	4	2	3	3	5	
data msg size (F)	3	3	3	2	3	2	3	
data msg size (D)	4	2	1	0	0	1	2	
RTT value: 1	3	3	3	2	3	2	3	2.714

Table 21 Test M with 4 dynamic channels (44% of the total channels) using Fair Distribution scheduling algorithm

Table 22 represents Test N in which five dynamic channels (50% of the total channels) are used and the scheduling algorithm is the Fair Distribution. Now there are five dynamic channels and the three data messages competing for them in time period 0. Data message “a” still gets the larger portion of dynamic channels, but now data message “f” gets one dynamic channel as well as “h” in time period 1. Again, the average time has decreased from the previous test.

```

CA-bothScenarios.txt  TEST #1
FairDistribution

time period | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
-----|-----|-----|-----|-----|-----|-----|-----|
F channel 1 | a0 | a1 | a5 |  |  |  |  |
F channel 2 |  | b0 | b1 | b4 |  |  |  |
F channel 3 | h0 | h1 | h3 |  |  | d0 | d1 |
F channel 4 | f0 | f1 |  | e0 | e1 |  |  |
F channel 5 |  | i0 | i1 | i4 |  |  |  |
D channel 6 |  | a2 | a6 |  | e2 |  |  |
D channel 7 |  | a3 | b2 |  |  |  |  |
D channel 8 |  | a4 | b3 |  |  |  |  |
D channel 9 |  | h2 | i2 |  |  |  |  |
D channel 10 |  | f2 | i3 |  |  |  |  |
-----|-----|-----|-----|-----|-----|-----|
# channels used | 3 | 10 | 9 | 3 | 2 | 1 | 1 |
bw utilization | 30% | 100% | 90% | 30% | 20% | 10% | 10% |

data message prefix | a | b | h | d | f | e | i | avg
-----|-----|-----|-----|-----|-----|-----|-----|
data message size | 7 | 5 | 4 | 2 | 3 | 3 | 5 |
data msg size (F) | 3 | 3 | 3 | 2 | 2 | 2 | 3 |
data msg size (D) | 4 | 2 | 1 | 0 | 1 | 1 | 2 |
RTT value: 1 | 3 | 3 | 3 | 2 | 2 | 2 | 3 | 2.571
-----|-----|-----|-----|-----|-----|-----|

```

Table 22 Test N with 5 dynamic channels (50% of the total channels) using Fair Distribution scheduling algorithm

Table 23 represents Test P in which ten dynamic channels (66% of the total channels) are used and the scheduling algorithm is the Fair Distribution. Now there are ten dynamic channels and the three data messages competing for them in time period 0. Data message “a” still gets the larger portion of dynamic channels, and “f” and “h” now get their proportional share. With more dynamic channels the proportional allocation is more evident. Again, the average time has decreased from the previous test.

```
CA-bothScenarios.txt TEST #1
FairDistribution
```

time period	0	1	2	3	4	5	6	
F channel 1	a0	a1						
F channel 2		b0	b1					
F channel 3	h0	h1				d0	d1	
F channel 4	f0	f1		e0	e1			
F channel 5		i0	i1					
D channel 6		a2	b2		e2			
D channel 7		a3	b3					
D channel 8		a4	b4					
D channel 9		a5	i2					
D channel 10		a6	i3					
D channel 11		h2	i4					
D channel 12		h3						
D channel 13		f2						
D channel 14								
D channel 15								
# channels used	3	13	8	1	2	1	1	
bw utilization	20%	87%	53%	7%	13%	7%	7%	
data message prefix	a	b	h	d	f	e	i	avg
data message size	7	5	4	2	3	3	5	
data msg size (F)	2	2	2	2	2	2	2	
data msg size (D)	5	3	2	0	1	1	3	
RTT value: 1	2	2	2	2	2	2	2	2.0

Table 23 Test P with 10 dynamic channels (66% of the total channels) using Fair Distribution scheduling algorithm

Table 24 represents Test Q in which 15 dynamic channels (75% of the total channels) are used and the scheduling algorithm is the Fair Distribution. Now there are 15 dynamic channels. However, since the data messages were not long enough to request additional dynamic channels, as in the case of time period 1, six dynamic channels go unused. As a result, the average time did not change from the previous test.

CA-bothScenarios.txt TEST #1
FairDistribution

time period	0	1	2	3	4	5	6	
F channel 1	a0	a1						
F channel 2		b0	b1					
F channel 3	h0	h1				d0	d1	
F channel 4	f0	f1		e0	e1			
F channel 5		i0	i1					
D channel 6		a2	b2		e2			
D channel 7		a3	b3					
D channel 8		a4	b4					
D channel 9		a5	i2					
D channel 10		a6	i3					
D channel 11		h2	i4					
D channel 12		h3						
D channel 13		f2						
D channel 14								
D channel 15								
D channel 16								
D channel 17								
D channel 18								
D channel 19								
D channel 20								
# channels used	3	13	8	1	2	1	1	
bw utilization	15%	65%	40%	5%	10%	5%	5%	
data message prefix	a	b	h	d	f	e	i	avg
data message size	7	5	4	2	3	3	5	
data msg size (F)	2	2	2	2	2	2	2	
data msg size (D)	5	3	2	0	1	1	3	
RTT value: 1	2	2	2	2	2	2	2	2.0

Table 24 Test Q with 15 dynamic channels (75% of the total channels) using Fair Distribution scheduling algorithm

The next set of tests run with the same parameters as before but with two changes. The first is the time period number is changed from 7 to 500. The second is randomness is used to generate message traffic. Examining the free time periods for each channel and generating a message block, again a random size, with a random start point, create the randomness. The randomness created here favored a light traffic load with an average utilization of 28%. Table 25 summarizes the data gathered from these tests.

test	scheduling algorithm	number of fix channels	number of dynamic channels	avg time periods	ratio of fix channels to total channels	portion of the bandwidth	delivery time	change from the base case (none)
1-30	none	5	0	149.216	1.0	1.00	298.43	0.00%
1-30	FirstComeFirstServe	5	1	110.633	1.2	0.83	243.39	18.44%
1-30	FirstComeFirstServe	5	2	89.882	1.4	0.71	215.72	27.72%
1-30	FirstComeFirstServe	5	3	76.078	1.6	0.63	197.80	33.72%
1-30	FirstComeFirstServe	5	4	65.581	1.8	0.56	183.63	38.47%
1-30	FirstComeFirstServe	5	5	57.761	2.0	0.50	173.28	41.94%
1-30	FirstComeFirstServe	5	10	35.563	3.0	0.33	142.25	52.33%
1-30	FirstComeFirstServe	5	15	25.462	4.0	0.25	127.31	57.34%
1-30	FairDistribution	5	1	115.680	1.2	0.83	254.50	14.72%
1-30	FairDistribution	5	2	98.524	1.4	0.71	236.46	20.77%
1-30	FairDistribution	5	3	86.704	1.6	0.63	225.43	24.46%
1-30	FairDistribution	5	4	77.866	1.8	0.56	218.02	26.94%
1-30	FairDistribution	5	5	70.307	2.0	0.50	210.92	29.32%
1-30	FairDistribution	5	10	47.703	3.0	0.33	190.81	36.06%
1-30	FairDistribution	5	15	35.965	4.0	0.25	179.83	39.74%

Table 25 Summary of 30 tests with a light traffic load

The next set of tests run with the same parameters as summarized in Table 25 with one change. The randomness created here favored a moderate traffic load with an average utilization of 72%. The data gathered from these tests are shown in Table 26.

test	scheduling algorithm	number of fix channels	number of dynamic channels	avg time periods	ratio of fix channels to total channels	portion of the bandwidth	delivery time	change from the base case (none)
1-30	none	5	0	105.234	1.0	1.00	210.47	0.00%
1-30	FirstComeFirstServe	5	1	84.138	1.2	0.83	185.10	12.05%
1-30	FirstComeFirstServe	5	2	66.987	1.4	0.71	160.77	23.61%
1-30	FirstComeFirstServe	5	3	53.727	1.6	0.63	139.69	33.63%
1-30	FirstComeFirstServe	5	4	43.968	1.8	0.56	123.11	41.51%
1-30	FirstComeFirstServe	5	5	36.695	2.0	0.50	110.09	47.69%
1-30	FirstComeFirstServe	5	10	20.175	3.0	0.33	80.70	61.66%
1-30	FirstComeFirstServe	5	15	14.164	4.0	0.25	70.82	66.35%
1-30	FairDistribution	5	1	85.144	1.2	0.83	187.32	11.00%
1-30	FairDistribution	5	2	70.094	1.4	0.71	168.22	20.07%
1-30	FairDistribution	5	3	57.656	1.6	0.63	149.90	28.78%
1-30	FairDistribution	5	4	48.754	1.8	0.56	136.51	35.14%
1-30	FairDistribution	5	5	42.217	2.0	0.50	126.65	39.82%
1-30	FairDistribution	5	10	24.901	3.0	0.33	99.60	52.68%
1-30	FairDistribution	5	15	17.681	4.0	0.25	88.41	57.99%

Table 26 Summary of 30 tests with a moderate traffic load

The final sets of tests run are the same parameters as summarized in Tables 25 and 26 with one change. The randomness created here favored a heavy traffic load with an average utilization of 92%. The data gathered from these tests are shown in Table 27. The data gathered from Tables 25, 26, and 27 are analyzed in the next section.

test	scheduling algorithm	number of fix channels	number of dynamic channels	avg time periods	ratio of fix channels to total channels	portion of the bandwidth	delivery time	change from the base case (none)
1-30	none	5	0	72.402	1.0	1.00	144.80	0.00%
1-30	FirstComeFirstServe	5	1	58.055	1.2	0.83	127.72	11.80%
1-30	FirstComeFirstServe	5	2	45.426	1.4	0.71	109.02	24.71%
1-30	FirstComeFirstServe	5	3	35.927	1.6	0.63	93.41	35.49%
1-30	FirstComeFirstServe	5	4	29.387	1.8	0.56	82.28	43.18%
1-30	FirstComeFirstServe	5	5	24.665	2.0	0.50	74.00	48.90%
1-30	FirstComeFirstServe	5	10	13.957	3.0	0.33	55.83	61.45%
1-30	FirstComeFirstServe	5	15	10.034	4.0	0.25	50.17	65.35%
1-30	FairDistribution	5	1	58.353	1.2	0.83	128.38	11.34%
1-30	FairDistribution	5	2	47.618	1.4	0.71	114.28	21.08%
1-30	FairDistribution	5	3	39.145	1.6	0.63	101.78	29.71%
1-30	FairDistribution	5	4	33.136	1.8	0.56	92.78	35.93%
1-30	FairDistribution	5	5	28.748	2.0	0.50	86.24	40.44%
1-30	FairDistribution	5	10	17.110	3.0	0.33	68.44	52.74%
1-30	FairDistribution	5	15	12.485	4.0	0.25	62.43	56.89%

Table 27 Summary of 30 tests with a heavy traffic load

D. ANALYSIS

A test consists of fifteen subtests. One subtest is the base case from which the two scheduling algorithms are compared to. Each scheduling algorithm runs seven subtests from which the number of dynamic channels are changed. Figure 14 is a graph showing the average time periods taken for each subtests. The values are taken from Table 25. The base test is identified on the x-axis as 1. The curve on the graph indicates as the number of dynamic channels is increased the average time periods decrease. However, the greatest change appears initially from the base test. The two scheduling algorithms initially perform similar until a variation begins to increase around 2 and 3 (28% and 38% of dynamic channels).

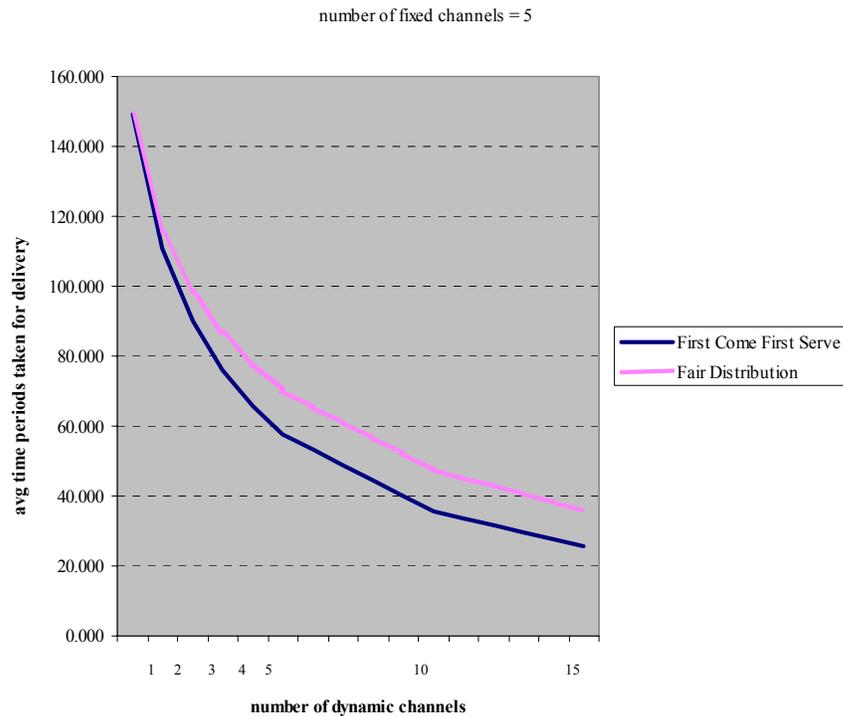


Figure 14 Average Time Periods Taken for Delivery from Table 25 (light load)

The amount of change in the delivery time is shown in Figure 15. The values are taken from Table 25, which ran a light traffic load. The base case is at 0% and the change using dynamic channels decreases. In this case the delivery time decreases most rapidly with the smaller ratio of fixed channels to total channels.

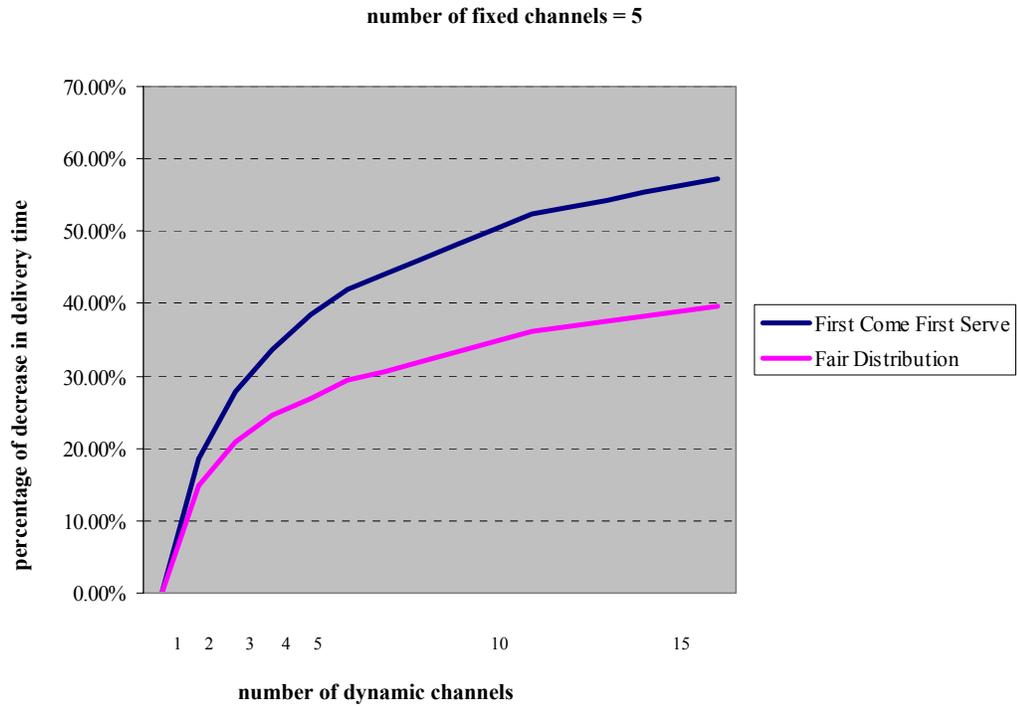


Figure 15 Decrease in Delivery Time from Table 25 (light load)

Figure 16 utilizes the 30 tests ran as summarized in Table 26. The average time periods taken for delivery increased from Figure 14 since the traffic load changed from a light load to a moderate load. However, the overall results are similar in that the greatest change occurs initially from the base test, and the curve is similar to Figure 14.

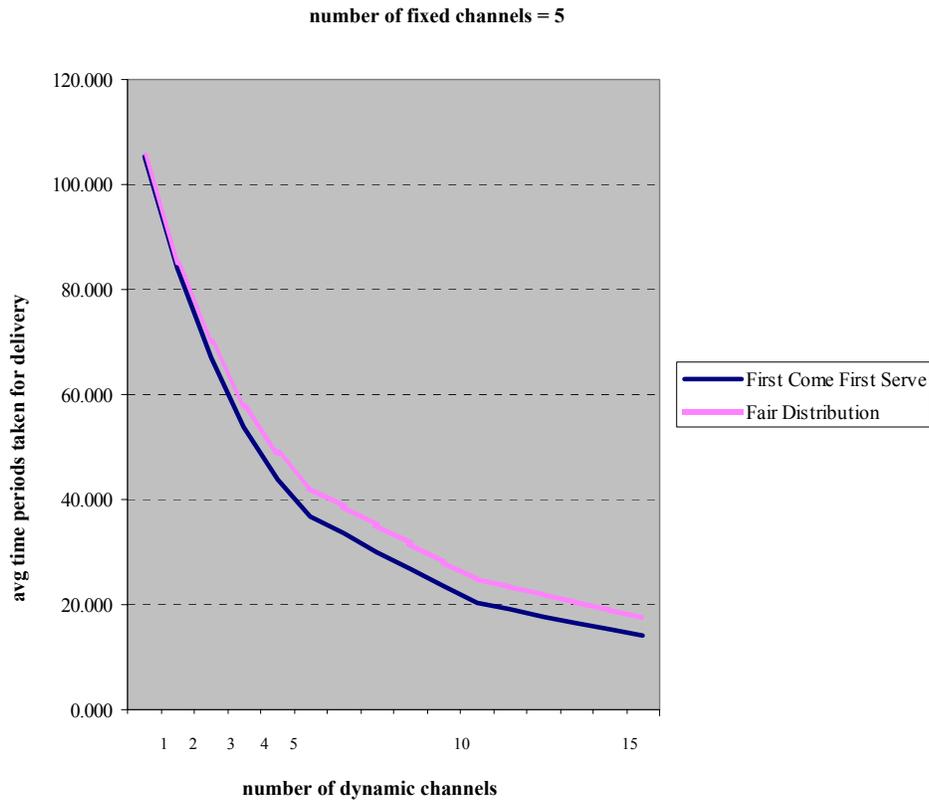


Figure 16 Average Time Periods Taken for Delivery from Table 26 (moderate load)

The change in delivery time is shown in Figure 17 using the same 30 tests as in Figure 16. The values are taken from Table 26. The base case is at 0% and the change using dynamic channels decreases. As in Figure 15 the delivery time decrease most rapidly with the smaller number of dynamic channels. However, the decrease is slightly more than in Figure 15 where the traffic load was light indicating that dynamic channels perform better with a moderate traffic load versus a light traffic load.

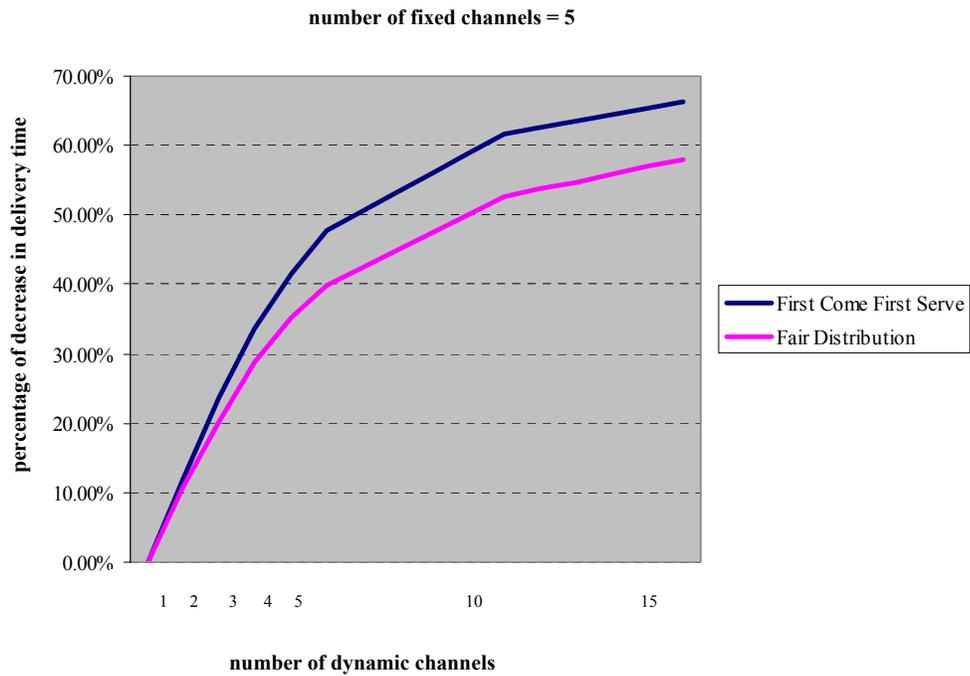


Figure 17 Decrease in Delivery Time from Table 26 (moderate load)

Figure 18 shows the average time periods taken for delivery from Table 27. In this case the traffic load was heavy which decreased the average time periods taken compared to Figures 14 and 16. The curve on the graph is similar to the curve in Figures 14 and 16. Again, as in the other two figures, the greatest change appears initially from the base test.

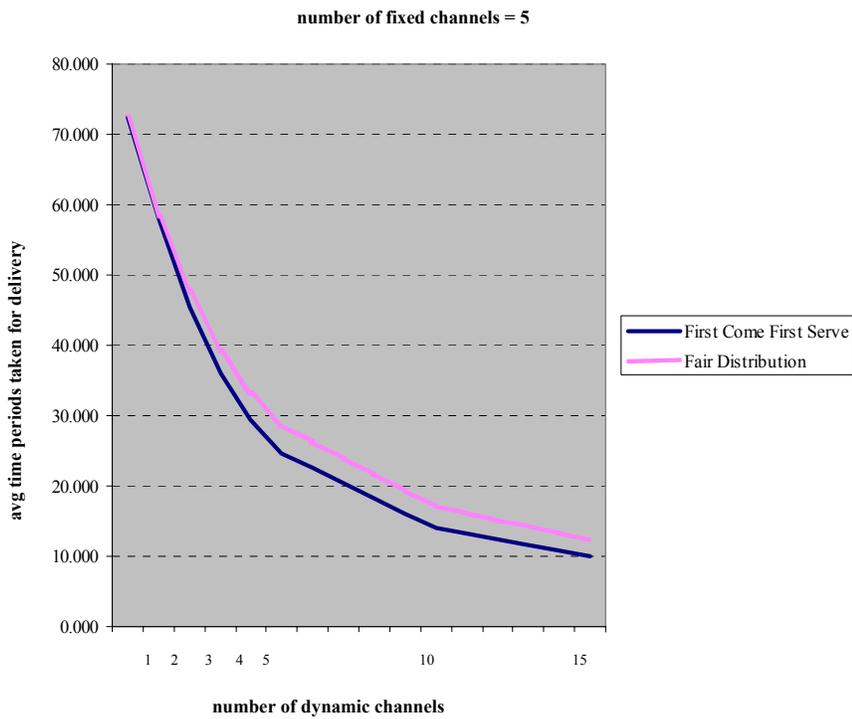


Figure 18 Average Time Periods Taken for Delivery from Table 27 (heavy load)

The final graph is shown in Figure 19. This graph shows the amount of change in the delivery time. The values are taken from Table 27, which ran a heavy traffic load. The base case is at 0% and the change using dynamic channels decreases. In this case the delivery time decreases most rapidly with the smaller ratio of fixed channels to total channels.

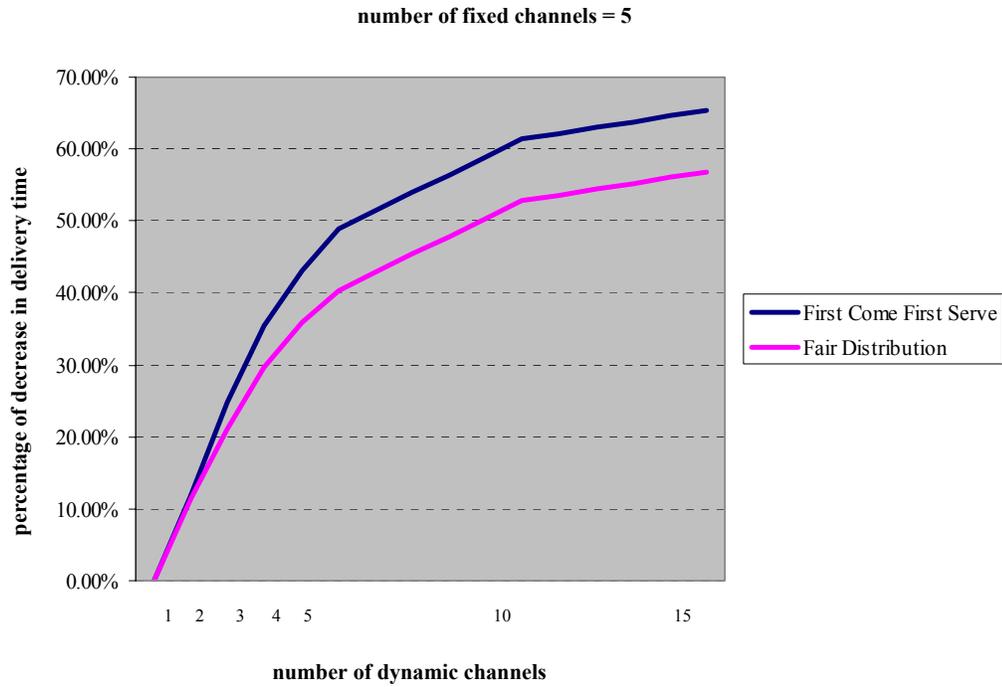


Figure 19 Decrease in Delivery Time from Table 27 (heavy load)

In comparing the graphs from Figures 14 through 19 there appears to be a relation in change occurring most dramatically with the lower ratio of fixed channels to total channels. The change begins to decrease beyond the ratio of 2 in which the number of dynamic channels is the same as the number of fixed channels. Therefore, if dynamic channels are utilized the most change is going to occur between a value 1 to 2 for the ratio of fixed channels to total channels. In addition, the tests utilized in Tables 26 and 27 simulated moderate to heavy data traffic on the fixed channels. If the data traffic were light in load then most likely dynamic channels would go unused thereby wasting bandwidth.

As the number of dynamic channels increases while the number of fixed channels remains constant, the two different scheduling algorithms perform slightly differently. In the case here, the First-Come-First-Serve scheduling algorithm performs better than the Fair Distribution in all three tests, with the light, moderate, and heavy traffic loads, respectively. In the First-Come-First-Serve scheduling algorithm, the first message using dynamic channels is able to complete its transmission in less time periods than in the Fair Distribution scheduling algorithm. The Fair Distribution algorithm will spread the message across more time periods as long as similar sized data messages are competing for dynamic channels, thereby increasing the number of time periods for transmission. For example, in Table 13 where five dynamic channels are used, data message “a” is able to complete its transmission in two time periods using the First-Come-First-Serve algorithm. In Table 22 where the same five dynamic channels are used, data message “a” completes its transmission in three time periods using the Fair Distribution algorithm. However, two other data messages, “h” and “f”, are able to begin using dynamic channels earlier under Fair Distribution. The overall delivery time average for all the data messages with Fair Distribution is slightly higher with 2.571 versus the average of 2.429 with First-Come-First-Serve scheduling algorithm.

E. POSSIBLE IMPLEMENTATIONS

Dynamic channel allocation has the potential to improve the transmission of data with moderate traffic loads. Fixed channel allocation becomes superior under higher traffic loads. Intuitively, it makes sense to dedicate fixed channels where there is continuous flow of data traffic. However, with data that isn't continuous or heavy but moderate the use of dynamic channels may reduce the time for transmission. This may be applied in an underwater acoustics network.

For example, an underwater acoustics network has several challenges. One of them is the propagation delay that is inherited with underwater communications. Another is the footprint of a station. Some stations may be hidden from another which renders carrier detection techniques ineffective. In fact, the likelihood of two nodes transmitting at the same time and colliding increases quite a bit. Therefore, the time to delivery data increases dramatically compared to a network operating in free space. Using dynamic channels may improve the delivery time.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSION

A. RECOMMENDATION

Dynamic channels possess the possibility of improving the delivery time. As networks continue to grow, more multimedia is desired, and time-sensitive applications increase, aggressive methods will be necessary to limit delivery latency. This is especially the case in wireless networks. Dynamic channel allocation is one such method. As previously mentioned DCA is preferable under moderate traffic loads.

B. FUTURE WORK

The model presented in this thesis is a demonstration of dynamic channel allocation. There are several enhancements that can be added to improve its performance and fidelity.

The propagation delay is included as part of the duration of each cell of the matrix. Future work may include modifying the matrix to account for both the propagation delay and processing time. Most important, it should be modified to support flow control mechanisms other than just Stop-and-Wait. Also, it may include congestion control.

The control of channels can be refined. The model presented here assumes a control station that allocates the channel. Implementing an allocation without a control station can be beneficial in networks with stations outside of a footprint. Additionally, the implemented model assumes a one-hop diameter network, with propagation delay uniform throughout. A hierarchical construct should be considered.

Other scheduling algorithms may be implemented. This thesis considered only first-come-first-served and fair distribution allocation scheduling. Others may better utilize the dynamic channels.

Data loss was assumed to be non-existence. However, this is not the case in real networks. Data loss can occur by many means such as channel fading, errors, and collisions. Future work may accommodate this.

Channels of varying size may reveal something interesting. This model used the same size channels, however, an array was created to hold the value of the channel capacity and the logic checked to make sure the data element could move into a dynamic channel of the same size. Examining the effects of different channel sizes to find a “best-mix of size” may hold merit.

Quality of service was not addressed in this thesis. Certain channels may have more priority than others. Various scheduling algorithms, such as priority queues, may allow for service qualities to be established. The Fair Distribution scenario provided an equal amount based on the total message size but a method may be added to better accommodate quality of service constraints, such as total latency.

VI. APPENDICES

A. PROGRAM - JAVA CLASS: CHANNEL ALLOCATION

```
/**
 * Filename: ChannelAllocation.java
 * Date: 18 April 2003
 * Revision: 5 September 2003
 * Author: Andy Kaminsky
 * Thesis: Channel Allocation
 * Compiler: Java2 SDK 1.4
 */

/**
 * The purpose of this class is to create and
 * initialize the arrays. This class calls upon
 * other classes for the scheduling algorithms
 * such as First-Come-First-Serve or FairDistribution,
 * for manipulating the allocation of dynamic
 * channels. This class also calls upon a display
 * class to show the output.
 *
 * @author: Andy Kaminsky
 */

/**
 * Assumption(s):
 * (1) For simulation purposes time to end is set
 * with the number of TIMEPERIODS
 * (2) Unique identifiers for sets of data blocks
 * are limited to the number of prefixIDs. In
 * addition, the prefixIDs are only the lower
 * and upper case of the alphabet and 3
 * characters long.
 * (3) This program currently utilizes the global
 * variable TIMEPERIODS = 13 and CHANNELS = 20.
 * Of course, these variables can be changed,
 * just make sure the set values fall into the
 * parameters. In addition, it is best to
 * recompile all the subclasses if the global
 * variables are changed
 */

import java.awt.*;
import java.util.*;
import java.io.*;
import javax.swing.*;
```

```

public class ChannelAllocation {

    //set global variables
    // *reminder to re-compile all classes if these
    // variables are changed

    //channels on the bandwidth
    public static final int CHANNELS    = 20;

    //number of time periods
    public static final int TIMEPERIODS = 13;

    //number of fixed channels
    public static final int FIXCHANNELS = 5;

    //identifiers for each channel
    public static final int SETTINGS    = 2;

    //abbreviation to denote fixed
    public static final String FIXCHANNELID = "F";

    //abbreviation to denote dynamic
    public static final String DYNCHANNELID = "D";

    public ChannelAllocation(){}

    public static void main(String[] args) {

        ChannelAllocation myChannelAllocation;
        myChannelAllocation = new ChannelAllocation();
        myChannelAllocation.start();

    } //end main

    //-----
    /**
     * The purpose of this method is to begin the program by
     * creating and initializing the Channel and Time Period
     * matrix.
     */

    public void start(){

        String fileName          = "";
        String filePrefix        = "";
        String fileOrder         = "";
        String fileExtension     = ".txt";
        String scheduleAlgorithm = "";
        int testCountDisplay     = 1;
    }
}

```

```

int numberDynamicChannels = 0;

//create 2 dimensional array to assign data in a
//matrix of channels and time periods
String [][] ChannelTimePeriodMatrix;
ChannelTimePeriodMatrix =
    new String [CHANNELS][TIMEPERIODS];

//create 2 dimensional array to backup original
//matrix for use in different scheduling
//algorithms
String [][] ChannelTimePeriodMatrixOriginal;
ChannelTimePeriodMatrixOriginal =
    new String [CHANNELS][TIMEPERIODS];

//initialize all values in array to null
initializeChannelTimePeriodMatrix(
    ChannelTimePeriodMatrix);

initializeChannelTimePeriodMatrix(
    ChannelTimePeriodMatrixOriginal);

//create 2 dimensional array
//assign fixed and dynamic channels in row [0]
//empty in row [1] but left for future changes
String [][] assignedChannel;
assignedChannel =
    new String [CHANNELS][SETTINGS];

//initialize all values in array to null
initializeAssignedChannel(assignedChannel);

//create a vector to store the original
//transfer time of a data block to make a
//comparison when the matrix is modified. For
//now the size is set to TIMEPERIODS; later
//this should be reduced to the number of
//data block prefixes for efficiency
int [] originalTransferTime;
originalTransferTime =
    new int [TIMEPERIODS*CHANNELS];

//initialize all values in array to zero
initializeOriginalTransferTime(
    originalTransferTime);

//create the set of prefix identifiers for a
//block of data
String [] prefixIDs =
    {"a","b","c","d","e","f","g","h","i","j","k",
     "l","m","n","o","p","q","r","s","t","u","v",
     "x","y","z","A","B","C","D","E","F","G","H",
     "I","J","K","L","M","N","O","P","Q","R","S",
     "T","U","V","X","Y","Z","aa","bb","cc","dd",

```

```

"ee", "ff", "gg", "hh", "ii", "jj", "kk", "ll", "mm",
"nn", "oo", "pp", "qq", "rr", "ss", "tt", "uu", "vv",
"ww", "xx", "yy", "zz", "AA", "BB", "CC", "DD", "EE",
"FF", "GG", "HH", "II", "JJ", "KK", "LL", "MM", "NN",
"OO", "PP", "QQ", "RR", "SS", "TT", "UU", "VV", "WW",
"XX", "YY", "ZZ", "ab", "ac", "ad", "ae", "af", "ag",
"ah", "ai", "aj", "ak", "al", "am", "an", "ao", "ap",
"aq", "ar", "as", "at", "au", "av", "aw", "ax", "ay",
"az", "bb", "aaa", "bbb", "ccc", "ddd", "eee",
"fff", "ggg", "hhh", "iii", "jjj", "kkk", "lll",
"mmm", "nnn", "ooo", "ppp", "TTT", "UUU", "VVV",
"WWW", "XXX", "YYY", "ZZZ"};

//ask user input for either the program's set
//values or randomly generate some number
//(default to "YES" for random)
boolean random = true;
random = userGeneratesRandomNumbers(random);

//ask user input for append or overwrite
//existing file
//(default to "YES" for append)
boolean append = true;
append = userInputAppendFile(append);

//ask user input for type of scheduling
//algorithm to run
//(default to first schedule algorithm listed)
int selection = 0;
selection =
    userInputSchedulingAlgorithm(selection);

//ask user input for number of tests to run
//(default to 1)
int numberOfTest = 0;
numberOfTest =
    userInputNumberOfTests(numberOfTest);

//ask user input for value of RTT
//(default to 1)
int valueOfRTT = 0;
// valueOfRTT = userInputValueOfRTT(valueOfRTT);
valueOfRTT = 1;

//ask user input for value of T
//(default to 1)
int valueOfT = 0;
// valueOfT = userInputValueOfT(valueOfT);
valueOfT = 1;

switch(selection){

    case 0:

```

```

        System.exit(0);
        break;

case 1:

    while(numberOfTest != 0){

        //First Come First Serve
        scheduleAlgorithm =
            "FirstComeFirstServe";

        //create a name for an output file
        filePrefix      =
            "CA-FirstComeFirstServe";
        fileOrder       = "BEFORE";

        int ratioT = 1;

        //run before any dynamic channels
        //are used
        runBeforeTest(
            ChannelTimePeriodMatrix,
            ChannelTimePeriodMatrixOriginal,
            assignedChannel,
            prefixIDs,
            originalTransferTime,
            random,
            append,
            scheduleAlgorithm,
            numberDynamicChannels,
            fileName,
            filePrefix,
            fileOrder,
            fileExtension,
            ratioT,
            valueOfRTT,
            valueOfT,
            testCountDisplay);

        numberDynamicChannels = 1;

        while((numberDynamicChannels)<
            (CHANNELS)){

            //run with dynamic channels
            fileOrder      = "AFTER";

            ratioT =

ratioOfT(numberDynamicChannels);

            runTest(
                ChannelTimePeriodMatrix,

```

```

ChannelTimePeriodMatrixOriginal,
                                assignedChannel,
                                originalTransferTime,
                                append,
                                numberDynamicChannels,
                                fileName,
                                filePrefix,
                                fileOrder,
                                fileExtension,
                                scheduleAlgorithm,
                                ratioT,
                                valueOfRTT,
                                valueOfT,
                                testCountDisplay);

dynamic                          //change the number of
                                //channels
                                numberDynamicChannels =

changeNumberDynamicChannels(
numberDynamicChannels);
                                }

                                numberDynamicChannels = 0;

                                //initialize values for another
                                //test
                                initializeMatrixForAnotherTest(
                                    ChannelTimePeriodMatrix,
                                    ChannelTimePeriodMatrixOriginal,
                                    assignedChannel,
                                    originalTransferTime);

                                testCountDisplay =
                                    testCountDisplay +1;

                                numberOfTest = numberOfTest -1;

                                }

                                break;

case 2:

                                while(numberOfTest != 0){

                                    //Fair Distribution
                                    scheduleAlgorithm =
                                        "FairDistribution";

                                    //create a name for an output file
                                    filePrefix =

```

```

    "CA-Fair Distribution";
fileOrder = "BEFORE";

int ratioT = 1;

//run before any dynamic channels
//are used
runBeforeTest(
    ChannelTimePeriodMatrix,
    ChannelTimePeriodMatrixOriginal,
    assignedChannel,
    prefixIDs,
    originalTransferTime,
    random,
    append,
    scheduleAlgorithm,
    numberDynamicChannels,
    fileName,
    filePrefix,
    fileOrder,
    fileExtension,
    ratioT,
    valueOfRTT,
    valueOfT,
    testCountDisplay);

numberDynamicChannels = 1;

while((numberDynamicChannels)<
    (CHANNELS)){

    //run with dynamic channels
    fileOrder = "AFTER";

    ratioT =

ratioOfT(numberDynamicChannels);

    runTest(
        ChannelTimePeriodMatrix,

ChannelTimePeriodMatrixOriginal,

        assignedChannel,
        originalTransferTime,
        append,
        numberDynamicChannels,
        fileName,
        filePrefix,
        fileOrder,
        fileExtension,
        scheduleAlgorithm,
        ratioT,
        valueOfRTT,
        valueOfT,
        testCountDisplay);

```

```

dynamic
//change the number of
//channels
numberDynamicChannels =

changeNumberDynamicChannels (
numberDynamicChannels);

}

numberDynamicChannels = 0;

//italize values for another test
initializeMatrixForAnotherTest(
    ChannelTimePeriodMatrix,
    ChannelTimePeriodMatrixOriginal,
    assignedChannel,
    originalTransferTime);

testCountDisplay =
    testCountDisplay +1;

numberOfTest = numberOfTest -1;

}

break;

case 3:

while(numberOfTest != 0){

//both scheduleAlgorithms
scheduleAlgorithm =
    "both scheduling algorithm";

//create a name for an output file
filePrefix = "CA-bothScenarios";
fileOrder = "BEFORE";

int ratioT = 1;

//run before any dynamic channels
//are used
runBeforeTest(
    ChannelTimePeriodMatrix,
    ChannelTimePeriodMatrixOriginal,
    assignedChannel,
    prefixIDs,
    originalTransferTime,
    random,
    append,
    scheduleAlgorithm,
    numberDynamicChannels,

```

```

        fileName,
        filePrefix,
        fileOrder,
        fileExtension,
        ratioT,
        valueOfRTT,
        valueOfT,
        testCountDisplay);

numberDynamicChannels = 1;

//run with dynamic channels
fileOrder = "AFTER";

while((numberDynamicChannels)<
      (CHANNELS)){

    //First Come First Serve
    scheduleAlgorithm =
        "FirstComeFirstServe";

    ratioT =

ratioOfT(numberDynamicChannels);

    runTest(
        ChannelTimePeriodMatrix,

ChannelTimePeriodMatrixOriginal,

        assignedChannel,
        originalTransferTime,
        append,
        numberDynamicChannels,
        fileName,
        filePrefix,
        fileOrder,
        fileExtension,
        scheduleAlgorithm,
        ratioT,
        valueOfRTT,
        valueOfT,
        testCountDisplay);

    //change the number of dynamic
    //channels
    numberDynamicChannels =

changeNumberDynamicChannels(
numberDynamicChannels);

}

numberDynamicChannels = 1;

while((numberDynamicChannels)<

```

```

(CHANNELS)) {

//Fair Distribution
scheduleAlgorithm =
    "FairDistribution";

ratioT =

ratioOfT(numberDynamicChannels);

runTest(
    ChannelTimePeriodMatrix,

ChannelTimePeriodMatrixOriginal,

    assignedChannel,
    originalTransferTime,
    append,
    numberDynamicChannels,
    fileName,
    filePrefix,
    fileOrder,
    fileExtension,
    scheduleAlgorithm,
    ratioT,
    valueOfRTT,
    valueOfT,
    testCountDisplay);

//change the number of

dynamic //channels
numberDynamicChannels =

changeNumberDynamicChannels(

numberDynamicChannels);

}

numberDynamicChannels = 0;

//initialize values for another
//test
initializeMatrixForAnotherTest(
    ChannelTimePeriodMatrix,
    ChannelTimePeriodMatrixOriginal,
    assignedChannel,
    originalTransferTime);

testCountDisplay =
    testCountDisplay +1;

numberOfTest = numberOfTest -1;

}

```

```

        break;

        default:

            System.exit(0);
            break;
    }

    System.exit(0);

} //end start

//-----
/**
 * The purpose of this method is to ask the user
 * whether to use random or fixed numbers.
 * Input is through a GUI.
 *
 * @param random
 *
 * @return random
 */

private boolean userGeneratesRandomNumbers(
    boolean random){

    int response;

    response = JOptionPane.showConfirmDialog(null,
        "want RANDOM values generated");

    switch(response){

        //use random numbers
        case 0:
            random = true;
            break;

        //use numbers already set
        case 1:
            random = false;
            break;

        //user enters CANCEL
        case 2:
            System.exit(0);
            break;

        default:
            System.exit(0);
            break;
    }
}

```

```

        return random;

} //end userGeneratesRandomNumbers

//-----
/**
 * The purpose of this method is to ask the user
 * whether the output created should be appended to
 * an existing file or overwrite the file (if one
 * exists).
 * Input is through a GUI.
 *
 * @param append
 *
 * @return append
 */

private boolean userInputAppendFile(
        boolean append){

    int response;

    response = JOptionPane.showConfirmDialog(null,
            "want to APPEND output to file");

    switch(response){

        //append to existing file
        case 0:
            append = true;
            break;

        //overwrite existing file
        case 1:
            append = false;
            break;

        //user enters CANCEL
        case 2:
            System.exit(0);
            break;

        default:
            System.exit(0);
            break;
    }

    return append;

} //end userInputAppendFile

//-----
/**
 * The purpose of this method is to ask the user

```

```

* which scheduling algorithm to run. Additional
* algorithms can be added in the future.
* Input is through a GUI.
*
* @param selection
*
* @return selection
*/

private int userInputSchedulingAlgorithm(
    int selection){

    String [] choices = {"First Come First Serve",
                        "Fair Distribution",
                        "both"};

    String input =
        (String)JOptionPane.showInputDialog(null,
        "choose scheduling algorithm",
        "Type of scheduling algorithm",
        JOptionPane.QUESTION_MESSAGE,null,
        choices,choices[0]);

    if (input == "First Come First Serve"){

        selection = 1;
    }

    if (input == "Fair Distribution"){

        selection = 2;
    }

    if (input == "both"){

        selection = 3;
    }

    return selection;

}

}

//-----
/**
 * The purpose of this method is to calculate the
 * ratio of fixed channels to dynamic channels.
 * For example,
 * 5:5 is 1 (0% dynamic channels)
 * 5:6 is 1.2 (16% dynamic channels)
 * 5:7 is 1.4 (28% dynamic channels)
 * 5:8 is 1.6 (38% dynamic channels)
 * 5:9 is 1.8 (44% dynamic channels)
 * 5:10 is 2 (50% dynamic channels)

```

```

*   5:15 is 3      (66% dynamic channels)
*   5:20 is 4      (75% dynamic channels)
*
*   @param numberDynamicChannels
*
*   @return ratioOfT
*/

private int ratioOfT(int numberDynamicChannels){

    double ratioT = 0;

    ratioT =
        (numberDynamicChannels + FIXCHANNELS)/
        FIXCHANNELS;

    return (int)ratioT;

} //end ratioOfT

//-----
/**
 * The purpose of this method is to ask the user
 * how many tests to run.
 * Input is through a GUI.
 *
 * @param numberOfTests
 *
 * @return numberOfTests
 */

private int userInputNumberOfTests(
    int numberOfTests){

    String input = JOptionPane.showInputDialog
        ("number of tests to run:");

    try {

        int value = Integer.parseInt(input);

        if (value == 0){

            numberOfTests = 0;

        }

        if (value >= 1){

            numberOfTests = value;

        }

    }

    //catches Number Format Exception error
    //conditions
    catch (NumberFormatException e){

```

```

        System.err.println ("not a valid number");
        System.exit(-1);

    }//end catch

    return numberOfTests;

}//end userInputNumberOfTests

//-----
/**
 * The purpose of this method is to ask the user
 * the RTT value
 * Input is through a GUI.
 *
 * @param valueOfRTT
 *
 * @return valueOfRTT
 */
private int userInputValueOfRTT(int valueOfRTT){

    String input = JOptionPane.showInputDialog
        ("value of RTT (1 to 999):");

    try {

        int value = Integer.parseInt(input);

        if (value == 0){

            valueOfRTT = 0;
        }

        if (value > 999){

            valueOfRTT = 1;
        }
        else {
            valueOfRTT = value;
        }
    }

    //catches Number Format Exception error
    //conditions
    catch (NumberFormatException e){

        System.err.println ("not a valid number");
        System.exit(-1);

    }//end catch

    return valueOfRTT;
}

```

```

} //end userInputValueOfRTT

//-----
/**
 * The purpose of this method is to ask the user
 * the T value
 * Input is through a GUI.
 *
 * @param valueOfT
 *
 * @return valueOfT
 */

private int userInputValueOfT(int valueOfT){

    String input = JOptionPane.showInputDialog
        ("value of T (1 to 999):");

    try {

        int value = Integer.parseInt(input);

        if (value == 0){

            valueOfT = 0;
        }

        if (value > 999){

            valueOfT = 1;
        }
        else {
            valueOfT = value;
        }
    }

    //catches Number Format Exception error
    //conditions
    catch (NumberFormatException e){

        System.err.println ("not a valid number");
        System.exit(-1);

    } //end catch

    return valueOfT;

} //end userInputValueOfT

//-----
/**
 * The purpose of this method is to change the
 * number of dynamic channels. Currently, the
 * program uses 1, 2, 3, 4, 5, 10, and 15.
 *

```

```

* @param numberDynamicChannels
*
* @return numberDynamicChannels
*/

private int changeNumberDynamicChannels(
    int numberDynamicChannels){

    //dynamic channels of 1,2,3,4
    if (numberDynamicChannels<5){
        numberDynamicChannels =
            numberDynamicChannels +1;
    }
    //dynamic channels of 5,10,15
    else {
        numberDynamicChannels =
            numberDynamicChannels +5;
    }

    return numberDynamicChannels;

}

}

//-----
/**
 * The purpose of this method is to create a name
 * for the output file
 *
 * @param filePrefix
 * @param fileOrder
 * @param fileExtension
 * @param testCountDisplay
 *
 * @return fileName
 */

private String createFileName(
    String filePrefix,
    String fileOrder,
    String fileExtension,
    int testCountDisplay){

    String fileName =
        filePrefix +
        // fileOrder +
        // " " +
        // Integer.toString(testCountDisplay)+
        // " " +
        fileExtension;

    return fileName;

}

}

```

```

//-----
/**
 * The purpose of this method is to initialize the
 * original transfer time matrix to zero values
 *
 * @param originalTransferTime
 */

private void initializeOriginalTransferTime(
    int [] originalTransferTime){

    for (int a = 0; a<TIMEPERIODS; a++){
        originalTransferTime[a]=0;
    }

}

}

//-----
/**
 * The purpose of this method is to initialize the
 * Channel and Time Period matrix to null values
 *
 * @param ChannelTimePeriodMatrix
 */

private void initializeChannelTimePeriodMatrix(
    String [][] ChannelTimePeriodMatrix){

    for(int a = 0; a<CHANNELS; a++){
        for(int b = 0; b<TIMEPERIODS; b++){
            ChannelTimePeriodMatrix[a][b] = null;
        }
    }

}

}

//-----
/**
 * The purpose of this method is to set some values
 * for representing data traffic in the Channel and
 * Time Period matrix. In addition, this methods
 * helps in troubleshooting the program.
 *
 * @param ChannelTimePeriodMatrix
 */

private void setSomeFixValuesChannelTimePeriodMatrix(
    String [][] ChannelTimePeriodMatrix){

    int dataSets = 0;

```

```

int channel          = 0;
int startTimePeriod = 0;
String name          = null;

//7 data sets to send on Channel 1 beginning at
//time period 0
dataSets            = 7;
channel             = 1;
startTimePeriod     = 0;
name                = "a";
setInitialData(ChannelTimePeriodMatrix,
                dataSet,
                channel,
                startTimePeriod,
                name);

//5 data sets to send on Channel 2 beginning at
//time period 1
dataSets            = 5;
channel             = 2;
startTimePeriod     = 1;
name                = "b";
setInitialData(ChannelTimePeriodMatrix,
                dataSet,
                channel,
                startTimePeriod,
                name);

//2 data sets to send on Channel 3 beginning at
//time period 5
dataSets            = 2;
channel             = 3;
startTimePeriod     = 5;
name                = "d";
setInitialData(ChannelTimePeriodMatrix,
                dataSet,
                channel,
                startTimePeriod,
                name);

//3 data sets to send on Channel 4 beginning at
//time period 2
dataSets            = 3;
channel             = 4;
startTimePeriod     = 3;
name                = "e";
setInitialData(ChannelTimePeriodMatrix,
                dataSet,
                channel,
                startTimePeriod,
                name);

//3 data sets to send on Channel 4 beginning at
//time period 0

```

```

dataSets      = 3;
channel       = 4;
startTimePeriod = 0;
name         = "f";
setInitialData(ChannelTimePeriodMatrix,
                dataSet,
                channel,
                startTimePeriod,
                name);

//4 data sets to send on Channel 3 beginning at
//time period 0
dataSets      = 4;
channel       = 3;
startTimePeriod = 0;
name         = "h";
setInitialData(ChannelTimePeriodMatrix,
                dataSet,
                channel,
                startTimePeriod,
                name);

//5 data sets to send on Channel 5 beginning at
//time period 1
dataSets      = 5;
channel       = 5;
startTimePeriod = 1;
name         = "i";
setInitialData(ChannelTimePeriodMatrix,
                dataSet,
                channel,
                startTimePeriod,
                name);

} //end setSomeValueChannelTimePeriodMatrix

//-----
/**
 * The purpose of this method is to set some random
 * values for representing data traffic in the
 * Channel and Time Period matrix
 *
 * @param ChannelTimePeriodMatrix
 * @param prefixIDs
 */

private void
setSomeRandomValuesChannelTimePeriodMatrix(
    String [][] ChannelTimePeriodMatrix,
    String [] prefixIDs){

    String name          = null;
    double myPercent     = 0;
    boolean exitCheck    = false;
    boolean needToGenerateID = true;

```

```

double targetUtilization = .4;
int maximumMessageGap = 2;
int minimumMessageSize = 5;

Vector usedPrefixIDs = new Vector(1,1);

Random percentCapacity = new Random();

int countOfFreeCells = TIMEPERIODS;

double threshold =
    ((double)countOfFreeCells/(double)TIMEPERIODS);

for(int a = 0; a<FIXCHANNELS; a++){

    while(threshold>targetUtilization){

        countOfFreeCells = 0;

        //count from the end to the beginning
        for(int z = TIMEPERIODS-1; z>1; z--){

            if (ChannelTimePeriodMatrix [a][z] ==
                null){

                countOfFreeCells =
                    countOfFreeCells +1;

            }

            else {
                z = 0;
            }

        }

        myPercent = percentCapacity.nextDouble();
        myPercent =
            Math.round(myPercent*countOfFreeCells);

        //get an unused prefix id
        while(!exitCheck){

            int stopChecking = 0;

            if (needToGenerateID){

                name =
                    prefixIDs[(int)
                        (prefixIDs.length*
                            Math.random())];

                needToGenerateID = false;

            }

        }

        for(int b=0; b<usedPrefixIDs.size());

```

```

        b++){
            if(usedPrefixIDs.get(b) == name){
                needToGenerateID = true;
                stopChecking = stopChecking
+1;
            }
        }
        if (stopChecking >1000){
            boolean response = false;
            JOptionPane.showConfirmDialog(
                null,
                "WARNING! Exhausted Prefix
List."
                +" Do you want to
continue?");

            if (response==false){
                System.exit(0);
            }
            else {
                response = true;
            }
        }
        if(!needToGenerateID){
            usedPrefixIDs.add(name);
            needToGenerateID = false;
            exitCheck = true;
        }
    }
    exitCheck = false;

    //generate start point
    int lastPointttoStart =
        (int) (countOfFreeCells - myPercent);

    //favor towards the front
    while (lastPointttoStart >
        maximumMessageGap){
        Random lastpoint = new Random();
        int myLast = 0;
        myLast =
            lastpoint.nextInt(
                lastPointttoStart);
        lastPointttoStart = myLast;
    }

```

```

Random startpoint = new Random();
int myStart = 0;
myStart =
    startpoint.nextInt(
        lastPointtoStart+1);
myStart =
    myStart+(TIMEPERIODS-countOfFreeCells);

//favor minimum block size
Random datalength = new Random();
int myDataLength = 0;
int maxAllowed = 0;
maxAllowed = (int)(myPercent);

if(maxAllowed >minimumMessageSize){

    while(myDataLength<
        minimumMessageSize){

        myDataLength =
            datalength.nextInt(
                maxAllowed+1);
    }

}
else {

    myDataLength =
        datalength.nextInt(maxAllowed+1);
}

//populate matrix
for(int c=0; c<myDataLength; c++){
    ChannelTimePeriodMatrix[a]
        [myStart+c]=name+c;
}

countOfFreeCells=0;

for(int z = TIMEPERIODS-1; z>1; z--){

    if (ChannelTimePeriodMatrix [a][z] ==
        null){
        countOfFreeCells =
            countOfFreeCells +1;
    }

    else {

        z = 0;
    }
}

threshold =

```

```

        ((double)countOfFreeCells/
         (double)TIMEPERIODS);

        myPercent          = 0;

    }//end while

    countOfFreeCells = TIMEPERIODS;

    threshold = ((double)countOfFreeCells/
                 (double)TIMEPERIODS);

    myPercent          = 0;

} // end for

} //end setSomeRandomValuesChannelTimePeriodMatrix

//-----
/**
 * The purpose of this method is to initialize the
 * Assigned Channel matrix to null values
 *
 * @param assignedChannel
 */

private void initializeAssignedChannel(
    String [][] assignedChannel){

    for(int a = 0; a<CHANNELS; a++){
        for(int b = 0; b<SETTINGS; b++){
            assignedChannel[a][b] = null;
        }
    }

} //end initializeAssignedChannel

//-----
/**
 * The purpose of this method is to set the
 * contents of the channel assignment (fixed or
 * dynamic). All channels are initially set to null
 * then overwritten with the type of channel
 *
 * @param assignedChannel
 * @param numberToAssign
 * @param typeOfChannel
 */

private void setAssignedChannel(
    String [][] assignedChannel,
    double numberToAssign,
    String typeOfChannel){

```

```

        double counter = numberToAssign;

        for(int a = 0; a<CHANNELS; a++){

            if(assignedChannel[a][0] == null){
                assignedChannel[a][0] = typeOfChannel;
                counter = counter -1;
            }

            if (counter == 0){
                a = CHANNELS;
            }

        }

    } //end setAssignedChannel

//-----
/**
 * The purpose of this method is to set initial
 * data
 *
 * @param ChannelTimePeriodMatrix
 * @param dataSets
 * @param channel
 * @param startTimePeriod
 * @param name
 */
private void setInitialData(
        String [][] ChannelTimePeriodMatrix,
        int dataSets,
        int channel,
        int startTimePeriod,
        String name){

    for(int a = 0; a<dataSets; a++){
        ChannelTimePeriodMatrix[channel-1]
            [a+startTimePeriod] = name+a;
    }

} //end setInitialData

//-----
/**
 * The purpose of this method is to copy the intial
 * values from the Channel and TimePeriod matrix to
 * an identical array so the values can be restored
 * in future tests
 *
 * @param ChannelTimePeriodMatrix
 * @param ChannelTimePeriodMatrixOriginal
 */
private void copyOriginalMatrix(
        String [][] ChannelTimePeriodMatrix,

```

```

        String [][] ChannelTimePeriodMatrixOriginal){

    for(int a = 0; a<CHANNELS; a++){

        for(int b = 0; b<TIMEPERIODS; b++){
            ChannelTimePeriodMatrixOriginal[a][b] =
                ChannelTimePeriodMatrix[a][b];
        }
    }

} //end copyOriginalMatrix

//-----
/**
 * The purpose of this method is to restore the
 * original values from the backup matrix
 *
 * @param ChannelTimePeriodMatrix
 * @param ChannelTimePeriodMatrixOriginal
 */

private void restoreOriginalMatrix(
    String [][] ChannelTimePeriodMatrix,
    String [][] ChannelTimePeriodMatrixOriginal){

    for(int a = 0; a<CHANNELS; a++){
        for(int b = 0; b<TIMEPERIODS; b++){
            ChannelTimePeriodMatrix[a][b] =
                ChannelTimePeriodMatrixOriginal[a][b];
        }
    }

} //end restoreOriginalMatrix

//-----
/**
 * The purpose of this method is to initialize the
 * matrix for another test using the same original
 * data
 *
 * @param ChannelTimePeriodMatrix
 * @param ChannelTimePeriodMatrixOriginal
 * @param assignedChannel
 * @param originalTransferTime
 */

private void initializeMatrixForAnotherTest(
    String [][] ChannelTimePeriodMatrix,
    String [][] ChannelTimePeriodMatrixOriginal,
    String [][] assignedChannel,
    int [] originalTransferTime){

    //initialize all values in array to null

```

```

        initializeChannelTimePeriodMatrix(
            ChannelTimePeriodMatrix);

        initializeChannelTimePeriodMatrix(
            ChannelTimePeriodMatrixOriginal);

        //initialize all values in array to null
        initializeAssignedChannel(assignedChannel);

        //initialize all values in array to zero
        initializeOriginalTransferTime(
            originalTransferTime);

} //end initializeMatrixForAnotherTest

//-----
/**
 * The purpose of this method is to run before any
 * algorithm begins.
 *
 * @param ChannelTimePeriodMatrix
 * @param ChannelTimePeriodMatrixOriginal
 * @param assignedChannel
 * @param prefixIDs
 * @param originalTransferTime
 * @param random
 * @param append
 * @param scheduleAlgorithm
 * @param numberDynamicChannels
 * @param fileName
 * @param filePrefix
 * @param fileOrder
 * @param fileExtension
 * @param ratioT
 * @param valueOfRTT
 * @param valueOfT
 * @param testCountDisplay
 */

private void runBeforeTest(
    String [][] ChannelTimePeriodMatrix,
    String [][] ChannelTimePeriodMatrixOriginal,
    String [][] assignedChannel,
    String [] prefixIDs,
    int [] originalTransferTime,
    boolean random,
    boolean append,
    String scheduleAlgorithm,
    int numberDynamicChannels,
    String fileName,
    String filePrefix,
    String fileOrder,
    String fileExtension,
    int ratioT,

```

```

        int valueOfRTT,
        int valueOfT,
        int testCountDisplay){

//create a file name
fileName = createFileName(filePrefix,
                           fileOrder,
                           fileExtension,
                           testCountDisplay);

//set the number of fixed channels
setAssignedChannel(assignedChannel,
                   FIXCHANNELS,
                   FIXCHANNELID);

if(random){
    setSomeRandomValuesChannelTimePeriodMatrix(
        ChannelTimePeriodMatrix,
        prefixIDs);
}
else {
    setSomeFixValuesChannelTimePeriodMatrix(
        ChannelTimePeriodMatrix);
}

DisplayChannel myBeforeTestOutput;
myBeforeTestOutput = new DisplayChannel();

myBeforeTestOutput.displayMatrix(
    ChannelTimePeriodMatrix,
    assignedChannel,
    originalTransferTime,
    scheduleAlgorithm,
    numberDynamicChannels,
    fileName,
    fileOrder,
    append,
    ratioT,
    valueOfRTT,
    valueOfT,
    testCountDisplay);

//initialize all values in array to null
initializeAssignedChannel(assignedChannel);

//copy original data
copyOriginalMatrix(
    ChannelTimePeriodMatrix,
    ChannelTimePeriodMatrixOriginal);

} //end runBeforeTest

//-----
/**

```

```

* The purpose of this method is to run the test.
*
* @param ChannelTimePeriodMatrix
* @param ChannelTimePeriodMatrixOriginal
* @param assignedChannel
* @param originalTransferTime
* @param append
* @param fileName
* @param filePrefix
* @param fileOrder
* @param fileExtension
* @param scheduleAlgorithm
* @param ratioT
* @param valueOfRTT
* @param valueOfT
* @param testCountDisplay
*/

private void runTest(
    String [][] ChannelTimePeriodMatrix,
    String [][] ChannelTimePeriodMatrixOriginal,
    String [][] assignedChannel,
    int [] originalTransferTime,
    boolean append,
    int numberDynamicChannels,
    String fileName,
    String filePrefix,
    String fileOrder,
    String fileExtension,
    String scheduleAlgorithm,
    int ratioT,
    int valueOfRTT,
    int valueOfT,
    int testCountDisplay){

    //create a file name for channel allocation
    fileName = createFileName(filePrefix,
                                fileOrder,
                                fileExtension,
                                testCountDisplay);

    //set the number of fixed channels
    setAssignedChannel(assignedChannel,
                        FIXCHANNELS,
                        FIXCHANNELID);

    //set the number of dynamic channels
    setAssignedChannel(assignedChannel,
                        numberDynamicChannels,
                        DYNCHANNELID);

    if (scheduleAlgorithm == "FairDistribution"){
        FairDistribution myTest;
        myTest = new FairDistribution();
    }
}

```

```

        myTest.fairDistribution(
            ChannelTimePeriodMatrix,
            assignedChannel);
    }

    if (scheduleAlgorithm == "FirstComeFirstServe"){
        FirstComeFirstServe myTest;
        myTest = new FirstComeFirstServe();
        myTest.firstComeFirstServe(
            ChannelTimePeriodMatrix,
            assignedChannel);
    }

    DisplayChannel myTestOutput;
    myTestOutput = new DisplayChannel();

    //display output
    myTestOutput.displayMatrix(
        ChannelTimePeriodMatrix,
        assignedChannel,
        originalTransferTime,
        scheduleAlgorithm,
        numberDynamicChannels,
        fileName,
        fileOrder,
        append,
        ratioT,
        valueOfRTT,
        valueOfT,
        testCountDisplay);

    //initialize all values in array to null
    initializeAssignedChannel(assignedChannel);

    //initialize all values in array to null
    initializeChannelTimePeriodMatrix(
        ChannelTimePeriodMatrix);

    //restore original data
    restoreOriginalMatrix(
        ChannelTimePeriodMatrix,
        ChannelTimePeriodMatrixOriginal);

} //end runTest

//-----
} //end class

```

B. PROGRAM – JAVA CLASS: DISPLAY CHANNEL

```
/**
 * Filename: DisplayChannel.java
 * Date: 23 May 2003
 * Revision: 5 September 2003
 * Author: Andy Kaminsky
 * Thesis: Channel Allocation
 * Compiler: Java2 SDK 1.4
 */

/**
 * The purpose of this class is to display the
 * matrix of channels and time periods.
 *
 * @author: Andy Kaminsky
 */

/**
 * Assumption(s):
 * (1) Output screen in system window
 * (2) Correct alignment is limited to 3 digits on
 * the time periods
 * (3) Cell size is fixed to 6 spaces
 * (4) Correct alignment for channel listing is
 * limited to 2 digits
 * (5) Correct alignment for bandwidth per channel
 * listing is limited to 3 digits
 */

/**
 * Note: There is a problem writing the -BEFORE
 * file when the user selects 'NO' for
 * appending. However, selecting 'YES'
 * creates the proper output.
 */

import java.util.*;
import java.io.*;

public class DisplayChannel
    extends ChannelAllocation {

    public DisplayChannel() {}

//-----
/**
 * The purpose of this method is to display the main
```

```

* Channel and Time Period matrix both to the screen
* and text file.
*
* @param ChannelTimePeriodMatrix
* @param assignedChannel
* @param originalTransferTime
* @param scenario
* @param numberDynamicChannels
* @param fileName
* @param fileOrder
* @param append
* @param ratioT
* @param valueOfRTT
* @param valueOfT
* @param testCountDisplay
*/

protected void displayMatrix(
    String [][] ChannelTimePeriodMatrix,
    String [][] assignedChannel,
    int [] originalTransferTime,
    String scheduleAlgorithm,
    int numberDynamicChannels,
    String fileName,
    String fileOrder,
    boolean append,
    int ratioT,
    int valueOfRTT,
    int valueOfT,
    int testCountDisplay){

    FileOutputStream fout;
    boolean printAssignedChannel    = false;
    double displayBreaks            = 0;
    double countCurrentBreaks       = 0;
    double desiredDisplayBreaks     = TIMEPERIODS;
    double overallChannelUtilization = 0;
    int beginFrom                   = 0;
    int endAt                       = 0;
    int leadingSpaces               = 0;
    int trailingSpaces              = 0;
    int cellSize                    = 6;
    int cellVariableSize            = 0;

    try{

        fout =
            new FileOutputStream (fileName,append);

        //print heading
        System.out.print(fileName+"  TEST #"+
            testCountDisplay+
            "\n"+"");
        new PrintStream(fout).println(fileName+

```

```

        " TEST #"+
            testCountDisplay);
System.out.println(scheduleAlgorithm);
new PrintStream(fout).println(
    scheduleAlgorithm);
new PrintStream(fout).println();
new PrintStream(fout).println();

displayBreaks =
    calculateDisplayBreaks(
        desiredDisplayBreaks);

while(displayBreaks != 0){

    //print time period heading
    System.out.print(
        " time period |");
    new PrintStream(fout).print(
        " time period |");

    beginFrom =
        calculateBeginFrom(
            countCurrentBreaks,
            desiredDisplayBreaks);

    endAt =
        calculateEndAt(
            countCurrentBreaks,
            desiredDisplayBreaks);

    if(endAt == TIMEPERIODS){

        displayBreaks = 0;

    }

    //print time period numbers
    for(double a = (beginFrom); a<(endAt);
        a++){

        leadingSpaces =
            calculateLeadingSpacesINT(
                (int)a,cellSize);

        trailingSpaces =
            calculateTrailingSpacesINT(
                (int)a,cellSize,leadingSpaces);

        for(int b = 0; b<leadingSpaces;
            b++){
            System.out.print(" ");
            new PrintStream(fout).print(
                " ");
        }
    }
}

```

```

        System.out.print((int) (a));
        new PrintStream(fout).print(
            (int) (a));

        for(int c = -1; c<trailingSpaces;
            c++){
            System.out.print(" ");
            new PrintStream(fout).print(
                " ");
        }

        System.out.print("|");
        new PrintStream(fout).print("|");
    }

    //print horizontal line
    new PrintStream(fout).println();
    System.out.print("\n-----");
    new PrintStream(fout).print(
        "-----");

    for(int d = 0; d<(endAt-beginFrom); d++){
        System.out.print("-----");
        new PrintStream(fout).print(
            "-----");
    }

    System.out.println();
    new PrintStream(fout).println();

    //print channel listing
    for(int e = 0; e<CHANNELS; e++){
        int tempChannel = e;

        if (assignedChannel[e][0] != null){
            printAssignedChannel = true;
        }

        while(printAssignedChannel){

            if (e<9 && e<CHANNELS){
                System.out.print(
                    assignedChannel[e][0]+
                    " channel "+(e+1)+
                    "    |");
                new PrintStream(fout).print(
                    assignedChannel[e][0]+
                    " channel "+(e+1)+
                    "    |");
            }

            else{
                System.out.print(
                    assignedChannel[e][0]+

```

```

        " channel "+(e+1)+"      |");
    new PrintStream(fout).print(
        assignedChannel[e][0]+
        " channel "+(e+1)+"      |");
}

// data in matrix
for(double f = (beginFrom);
    f<(endAt); f++){

    int g = (int)f;

    cellVariableSize =
        findCellVariableSize(
            ChannelTimePeriodMatrix,
            e,
            g,
            cellSize);

    leadingSpaces =
        calculateLeadingSpacesCHAR(
            cellVariableSize,
            cellSize);

    trailingSpaces =
        calculateTrailingSpacesCHAR(
            cellVariableSize,
            cellSize,
            leadingSpaces);

    //print empty cell in matrix
    if(ChannelTimePeriodMatrix[e][g] ==
        null){
        for(int h = 0;
            h<cellSize+1; h++){
            System.out.print(" ");
            new PrintStream(fout).print(
                " ");
        }
    }

    //print non-empty cell in matrix
    else {

        //print leading spaces
        for(int i=0;
            i<leadingSpaces; i++){
            System.out.print(" ");
            new PrintStream(fout).print(
                " ");
        }

        //print variable

```

```

        System.out.print(
            ChannelTimePeriodMatrix
            [e][g]);
        new PrintStream(fout).print(
            ChannelTimePeriodMatrix
            [e][g]);

        //print trailing spaces
        for(int j=0;
            j<trailingSpaces; j++){
            System.out.print(" ");
            new PrintStream(fout).print(
                " ");
        }
    }

    System.out.print("|");
    new PrintStream(fout).print("|");
}

System.out.println();
new PrintStream(fout).println();

printAssignedChannel = false;
}

} //end while

//print horizontal line
System.out.print("-----");
new PrintStream(fout).print(
    "-----");

for(int k = 0; k<(endAt-beginFrom); k++){
    System.out.print("-----");
    new PrintStream(fout).print("-----");
}

System.out.println();
new PrintStream(fout).println();

//channels used in each time period
System.out.print("# channels used | ");
new PrintStream(fout).print(
    "# channels used | ");

for(double l = (beginFrom); l<(endAt); l++){
    int m = (int)l;
    int tempTimePeriod = m;
    double numberChannelsUsedPerTimePeriod =
        0;

    numberChannelsUsedPerTimePeriod =
        calculateNumberChannelsUsedPerTimePeriod(

```

```

        ChannelTimePeriodMatrix,
        tempTimePeriod);

//align if channels used is 1 digit (0-9)
if(Math.round(
    numberChannelsUsedPerTimePeriod)<10){
    System.out.print(
        " "+Math.round(
            numberChannelsUsedPerTimePeriod)+
            " | ");
    new PrintStream(fout).print(
        " "+Math.round(
            numberChannelsUsedPerTimePeriod)+
            " | ");
    }

//align if channels used is 2 digits (10-99)
else {
    System.out.print(
        Math.round(
            numberChannelsUsedPerTimePeriod)+
            " | ");
    new PrintStream(fout).print(
        Math.round(
            numberChannelsUsedPerTimePeriod)+
            " | ");
    }
}

System.out.println();
new PrintStream(fout).println();

//bandwidth utilization
System.out.print("bw utilization | ");
new PrintStream(fout).print(
    "bw utilization | ");

for(double lll = (beginFrom); lll<(endAt);
    lll++){
    int mmm = (int)lll;
    int tempTimePeriod = mmm;
    double numberChannelsUsedPerTimePeriod = 0;
    double numberChannelsAssigned = 0;
    double bandwidthUtilization = 0;

    numberChannelsUsedPerTimePeriod =
        calculateNumberChannelsUsedPerTimePeriod(
            ChannelTimePeriodMatrix,
            tempTimePeriod);

    numberChannelsAssigned =
        calculateNumberChannelsAssigned(
            assignedChannel);

```

```

bandwidthUtilization =
    calculateBandwidthUtilization(
        numberChannelsAssigned,
        numberChannelsUsedPerTimePeriod);

//align if utilization is 1 digit (0%-9%)
if(Math.round(bandwidthUtilization)<10){
    System.out.print(
        " "+Math.round(
            bandwidthUtilization)+"% ");
    new PrintStream(fout).print(
        " "+Math.round(
            bandwidthUtilization)+"% ");
}

//align if utilization is 2 digits (10%-99%)
else {
    System.out.print(
        Math.round(
            bandwidthUtilization)+"% ");
    new PrintStream(fout).print(
        Math.round(
            bandwidthUtilization)+"% ");
}
//align if utilization is 3 digits (100%)
if(Math.round(bandwidthUtilization)<99){
    System.out.print(" | ");
    new PrintStream(fout).print(" | ");
}
else {
    System.out.print("| ");
    new PrintStream(fout).print("| ");
}
}

for(int n =0; n<2; n++){
    System.out.println();
    new PrintStream(fout).println();
}

//overall utilization
overallChannelUtilization =
    calculateOverallUtilization(
        ChannelTimePeriodMatrix);

/*
System.out.println(
    "overall utilization of all the "+
    "channels and time periods is "+
    overallChannelUtilization+"\n");

new PrintStream(fout).println(
    "overall utilization of "+

```

```

        "all the channels and "+
        "time periods is "+
        overallChannelUtilization+"%");
new PrintStream(fout).println(" ");
*/
    if(countCurrentBreaks == displayBreaks){
        displayBreaks = 0;
    }

    countCurrentBreaks = countCurrentBreaks+1;
    System.out.println();
    new PrintStream(fout).println();
}

//display delivery time
DisplayDelivery myOutputDelivery;
myOutputDelivery = new DisplayDelivery();

myOutputDelivery.delivery(
    ChannelTimePeriodMatrix,
    assignedChannel,
    originalTransferTime,
    scheduleAlgorithm,
    numberDynamicChannels,
    fileName,
    fileOrder,
    desiredDisplayBreaks,
    cellSize,
    ratioT,
    valueOfRTT,
    valueOfT,
    testCountDisplay);

for(int n =0; n<3; n++){
    System.out.println();
    new PrintStream(fout).println();
}

// Close the output stream
fout.close();
}

// Catches any error conditions
catch (IOException e){
    System.err.println (
        "unable to write to file");
    System.exit(-1);
} //end catch

} //end displayChannelTimePeriodMatrix

//-----
/**
 * The purpose of this method is to display the contents

```

```

* of the channel assignment (fixed or dynamic). This
* helps in troubleshooting.
*
* @param assignedChannel
*
*/

private void displayAssignedChannel(
    String [][] assignedChannel){

    for(int a = 0; a<CHANNELS; a++){
        System.out.println("Channel "+a+" is "+
            assignedChannel[a][0]);
    }

}

}

//-----
/**
 * The purpose of this method is to calculate current
 * channel utilization
 *
 * @param ChannelTimePeriodMatrix
 * @param tempChannel
 *
 * @return utilization
 */

private double calculateCurrentChannelUtilization(
    String [][] ChannelTimePeriodMatrix,
    int tempChannel){

    double utilization = 0;
    double timePeriodUsed = 0;

    for(int a = 0; a<TIMEPERIODS; a++){
        if(!(ChannelTimePeriodMatrix[tempChannel][a] ==
            null)){
            timePeriodUsed++;
        }
    }

    utilization = ((timePeriodUsed/TIMEPERIODS)*100);

    return utilization;

}

//-----
/**
 * The purpose of this method is to calculate current
 * time period utilization
 *
 * @param ChannelTimePeriodMatrix
 * @param tempTimePeriod

```

```

*
* @return utilization
*/

private double calculateCurrentTimePeriodUtilization(
    String [][] ChannelTimePeriodMatrix,
    int tempTimePeriod){

    double utilization = 0;
    double channelUsed = 0;

    for(int a = 0; a<CHANNELS; a++){
        if(!(ChannelTimePeriodMatrix[a][tempTimePeriod] ==
            null)){
            channelUsed++;
        }
    }

    utilization = ((channelUsed/CHANNELS)*100);

    return utilization;

} //end calculateCurrentTimePeriodUtilization

//-----
/**
 * The purpose of this method is to calculate overall
 * utilization. This also helps in checking that the
 * same value is returned in all the tests ran.
 *
 * @param ChannelTimePeriodMatrix
 *
 * @return utilization
 */

private double calculateOverallUtilization(
    String [][] ChannelTimePeriodMatrix){

    double utilization = 0;
    double timePeriodUsed = 0;

    for(int a = 0; a<TIMEPERIODS; a++){
        for(int b = 0; b<CHANNELS; b++){
            if(!(ChannelTimePeriodMatrix[b][a] ==
                null)){
                timePeriodUsed++;
            }
        }
    }

    utilization =
        (Math.round((timePeriodUsed/
            (TIMEPERIODS*CHANNELS))*100));

    return utilization;
}

```

```

    }//end calculateOverallUtilization

//-----
/**
 * The purpose of this method is to calculate the total
 * number of display breaks for the screen. This is
 * useful when all the time periods in the Channel Time
 * Period matrix cannot be properly displayed on a
 * particular screen.
 *
 * @param desiredDisplayBreaks
 *
 * @return displayBreaks
 */

private double calculateDisplayBreaks(
    double desiredDisplayBreaks){

    double displayBreaks = (int)desiredDisplayBreaks;
    displayBreaks = TIMEPERIODS/displayBreaks;

    return displayBreaks;

}//end calculateDisplayBreaks

//-----
/**
 * The purpose of this method is to calculate where the
 * beginning of the matrix is for each display break.
 * For example, if the desired display break is 7 then
 * the beginFrom value will be 0 for the first display,
 * 7 for the next, 14 for the next, and so on.
 *
 * @param countCurrentBreaks
 * @param desiredDisplayBreaks
 *
 * @return beginFrom
 */

private int calculateBeginFrom(
    double countCurrentBreaks,
    double desiredDisplayBreaks){

    int beginFrom = 0;

    if((countCurrentBreaks*desiredDisplayBreaks)>
        TIMEPERIODS){

        beginFrom =
            ((int)countCurrentBreaks*
            (int)desiredDisplayBreaks)-
            (((int)countCurrentBreaks*
            (int)desiredDisplayBreaks)-
            TIMEPERIODS);
    }
}

```

```

    }

    else{

        beginFrom =
            (int)countCurrentBreaks
            *(int)desiredDisplayBreaks;
    }

    return beginFrom;
}

} //end calculateBeginFrom

//-----
/**
 * The purpose of this method is to calculate where the
 * end of the matrix is for each display break. For
 * example, if the desired display break is 7 then the
 * endAt value will be 6 for the first display, 13 for
 * the next, 20 for the next, and so on.
 *
 * @param countCurrentBreaks
 * @param desiredDisplayBreaks
 *
 * @return endAt
 */

private int calculateEndAt(
    double countCurrentBreaks,
    double desiredDisplayBreaks){

    int endAt = 0;

    if((desiredDisplayBreaks*(countCurrentBreaks+1)
        >=TIMEPERIODS)){

        endAt = TIMEPERIODS;

    }

    else{

        endAt = ((int)desiredDisplayBreaks*
            ((int)countCurrentBreaks+1));

    }

    return endAt;

}

} //end calculateEndAt

//-----
/**
 * The purpose of this method is to calculate the
 * number of leading spaces to align the variable in
 * the cell.
 *

```

```

* @param cellVariable
* @param cellSize
*
* @return leadingSpaces
*/

private int calculateLeadingSpacesINT(
    int cellVariable,
    int cellSize){

    int digits = calculateDigits(cellVariable);
    int leadingSpaces = 0;
    int addTo = 0;

    leadingSpaces = cellSize - digits;

    if(leadingSpaces == 2){
        leadingSpaces = 3;
    }
    if(leadingSpaces == 4){
        leadingSpaces = 3;
    }
    if(leadingSpaces == 5){
        leadingSpaces = 2;
    }

    return leadingSpaces;
}

} //end calculateLeadingSpacesINT

//-----
/**
* The purpose of this method is to calculate the
* number of trailing spaces to align the variable in
* the cell.
*
* @param cellVariable
* @param cellSize
* @param leadingSpaces
*
* @return trailingSpaces
*/

private int calculateTrailingSpacesINT(
    int cellVariable,
    int cellSize,
    int leadingSpaces){

    int digits = calculateDigits(cellVariable);
    int trailingSpaces = 0;
    int addTo = 0;

    trailingSpaces = cellSize - digits - leadingSpaces;

```

```

        return trailingSpaces;
    } //end calculateTrailingSpacesINT

    //-----
    /**
     * The purpose of this method is to calculate the
     * number of digits in a number.
     *
     * @param cellVariable
     *
     * @return digits
     */

    private int calculateDigits(int cellVariable){

        int digits = 1;

        if(cellVariable>=10){
            digits = 2;
            if(cellVariable>=100){
                digits = 3;
                if(cellVariable>=1000){
                    digits = 4;
                }
            }
        }

        return digits;
    } //end calculateDigits

    //-----
    /**
     * The purpose of this method is to calculate the
     * number of characters in the cell.
     *
     * @param ChannelTimePeriodMatrix
     * @param currentChannel
     * @param currentTimePeriod
     * @param cellSize
     *
     * @return size
     */

    private int findCellVariableSize(
        String[][]ChannelTimePeriodMatrix,
        int currentChannel,
        int currentTimePeriod,
        int cellSize){

        int size = 0;

        if (ChannelTimePeriodMatrix[currentChannel]
            [currentTimePeriod]!=null){

```

```

        size =
            ChannelTimePeriodMatrix[currentChannel]
                [currentTimePeriod].length();
    }

    else {
        size = cellSize;}

    return size;

} //end findCellVariableSize

//-----
/**
 * The purpose of this method is to calculate the
 * number of leading spaces to align the variable in
 * the cell.
 *
 * @param cellVariableSize
 * @param cellSize
 *
 * @return leadingSpaces
 */

private int calculateLeadingSpacesCHAR(
    int cellVariableSize,
    int cellSize){

    int leadingSpaces = cellSize - cellVariableSize;

    if(leadingSpaces==4){
        leadingSpaces=3;
    }
    if(leadingSpaces==3){
        leadingSpaces=3;
    }
    if(leadingSpaces==2){
        leadingSpaces=2;
    }
    if(leadingSpaces==1){
        leadingSpaces=2;
    }
    if(leadingSpaces==0){
        leadingSpaces=1;
    }

    return leadingSpaces;

} //end calculateLeadingSpacesCHAR

//-----
/**
 * The purpose of this method is to calculate the
 * number of trailing spaces to align the variable in

```

```

* the cell.
*
* @param cellVariableSize
* @param cellSize
* @param leadingSpaces
*
* @return trailingSpaces
*/

private int calculateTrailingSpacesCHAR(
    int cellVariableSize,
    int cellSize,
    int leadingSpaces){

    int trailingSpaces = cellSize - cellVariableSize -
        leadingSpaces +1;

    return trailingSpaces;

} //end calculateTrailingSpacesCHAR

//-----
/**
* The purpose of this method is to calculate the
* number of channels used during a time period
*
* @param ChannelTimePeriodMatrix
* @param tempTimePeriod
*
* @return channelsUsed
*/

private double calculateNumberChannelsUsedPerTimePeriod(
    String [][] ChannelTimePeriodMatrix,
    int tempTimePeriod){

    double channelsUsed = 0;

    for(int a = 0; a<CHANNELS; a++){
        if(!(ChannelTimePeriodMatrix[a]
            [tempTimePeriod] == null)){
            channelsUsed++;
        }
    }

    return channelsUsed;

} //end calculateNumberChannelsUsedPerTimePeriod

//-----
/**
* The purpose of this method is to calculate the
* number of channels assigned either as fixed or
* dynamic
*

```

```

* @param assignedChannel
*
* @return numberChannelsAssigned
*/

private double calculateNumberChannelsAssigned(
    String [][] assignedChannel){

    double numberChannelsAssigned = 0;

    for(int a = 0; a<CHANNELS; a++){

        if(assignedChannel[a][0] != null){
            numberChannelsAssigned =
                numberChannelsAssigned +1;
        }

    }

    return numberChannelsAssigned;

} //end calculateNumberChannelsAssigned

//-----
/**
 * The purpose of this method is to calculate the
 * bandwidth utilization
 *
 * @param numberChannelsAssigned
 * @param numberChannelsUsedPerTimePeriod
 *
 * @return bandwidthUtilization
 */

private double calculateBandwidthUtilization(
    double numberChannelsAssigned,
    double numberChannelsUsedPerTimePeriod){

    double bandwidthUtilization =
        (numberChannelsUsedPerTimePeriod/
            numberChannelsAssigned)*100;

    return bandwidthUtilization;

} //end calculateBandwidthUtilization

//-----

} //end class

```

C. PROGRAM – JAVA CLASS: DISPLAY DELIVERY TIME

```
/**
 * Filename: DisplayThroughput.java
 * Date: 16 June 2003
 * Revision: 5 September 2003
 * Author: Andy Kaminsky
 * Thesis: Channel Allocation
 * Compiler: Java2 SDK 1.4
 */

/**
 * The purpose of this class is to calculate and
 * display the delivery time through the use of
 * dynamic channels.
 *
 * @author: Andy Kaminsky
 */

/**
 * Assumptions:
 * (1) The size of values in the array are around
 * 6 characters.
 * (2) For correction display of percentage the
 * prefixIDs are limited to three characters.
 */

import java.util.*;
import java.io.*;

public class DisplayDelivery
    extends DisplayChannel {

    public DisplayDelivery() {}

//-----
/**
 * The purpose of this method is to calculate and
 * display the delivery time through the use of dynamic
 * channels.
 *
 * @param ChannelTimePeriodMatrix
 * @param assignedChannel
 * @param originalTransferTime
 * @param scheduleAlgorithm
 * @param fileName
 * @param numberDynamicChannels
 * @param fileOrder
 */
```

```

* @param desiredDisplayBreaks
* @param cellSize
* @param ratioT
* @param valueOfRTT
* @param valueOfT
* @param testCountDisplay
*/

protected void delivery(
    String [][] ChannelTimePeriodMatrix,
    String [][] assignedChannel,
    int [] originalTransferTime,
    String scheduleAlgorithm,
    int numberDynamicChannels,
    String fileName,
    String fileOrder,
    double desiredDisplayBreaks,
    int cellSize,
    int ratioT,
    int valueOfRTT,
    int valueOfT,
    int testCountDisplay){

    String graphName = "CA-data-";
    String dataFileExport = "CA-dataExport.txt";
    char constructPrefix;
    int countIDs = 0;

    String graphFileName =
        dataFileName(fileName, graphName);

    //count dataID blocks
    String dataPrefix =
        (dataIdPrefix(ChannelTimePeriodMatrix));

    for(int a=0; a<dataPrefix.length(); a++){
        constructPrefix = dataPrefix.charAt(a);

        if(constructPrefix == '['){
            countIDs = countIDs+1;
        }
    }

    //create one dimensional array to store:
    // [0] data prefix identifier
    String [] delayDataID;
    delayDataID = new String [countIDs];

    //store data identifiers (prefix) into array
    storeDataIds(delayDataID, dataPrefix, countIDs);

    //create two dimensional array to store:
    // [0] data block size
    // [1] data block size of fixed
    // [2] data block size of dynamic

```

```

// [3] RTT
// [4] T
// [5] RTT+T
int [][] delayDataNumber;
delayDataNumber = new int [6][countIDs];

//count data block size and store in array
countStoreDataBlockSize(
    ChannelTimePeriodMatrix,
    delayDataID,
    delayDataNumber,
    countIDs);

//count time periods taken to send (RTT)
countStoreRTT(ChannelTimePeriodMatrix,
    delayDataID,
    delayDataNumber,
    countIDs,
    valueOfRTT);

//calculate T
calculateStoreT(delayDataNumber,
    countIDs,
    ratioT,
    valueOfT);

//calculate delivery time (RTT+T)
calculateDeliveryTime(
    delayDataNumber,
    countIDs,
    originalTransferTime,
    fileOrder);

//display the time delivery matrix
displayTimeDeliveryMatrix(
    delayDataID,
    delayDataNumber,
    countIDs,
    fileName,
    valueOfRTT,
    valueOfT,
    scheduleAlgorithm,
    numberDynamicChannels,
    dataFileExport,
    fileOrder,
    testCountDisplay);

/*
//calculate and display delivery change
calculateDisplayThroughputChange(
    delayDataID,
    delayDataNumber,
    originalTransferTime,
    countIDs,
    desiredDisplayBreaks,

```

```

        cellSize,
        fileName,
        graphFileName);
        */

} //end delivery

//-----
/**
 * The purpose of this method is to add a graph name
 * to the existing file name
 *
 * @param fileName,
 * @param graphName
 *
 * @return name
 */

private String dataFileName(String fileName,
                           String graphName){

    String name = graphName + fileName;

    return name;

} //end dataFileName

//-----
/**
 * The purpose of this method is to find all the data
 * block identifiers (the prefix) in the
 * ChannelTimePeriod matrix
 *
 * @param ChannelTimePeriodMatrix
 *
 * @return dataIDs
 */

private String dataIdPrefix(
    String [][] ChannelTimePeriodMatrix){

    String tempID    = "";
    String workValue = "";
    String dataIDs   = " ";
    char   tempChar;

    Vector usedIDs = new Vector(1,1);

    for(int a=0; a<CHANNELS; a++){

        for(int b=0; b<TIMEPERIODS; b++){

            if(ChannelTimePeriodMatrix[a]
                [b] != null){

```



```

* The purpose of this method is to display the
* contents of the delayDataID matrix and the
* delayDataNumber matrix
*
* @param delayDataID
* @param delayDataNumber
* @param countIDs
* @param fileName
* @param valueOfRTT
* @param valueOfT
* @param scheduleAlgorithm
* @param fileName
* @param dataFileExport
* @param fileOrder
* @param testCountDisplay
*/

private void displayTimeDilveryMatrix(
    String [] delayDataID,
    int [][] delayDataNumber,
    int countIDs,
    String fileName,
    int valueOfRTT,
    int valueOfT,
    String scheduleAlgorithm,
    int numberDynamicChannels,
    String dataFileExport,
    String fileOrder,
    int testCountDisplay){

    FileOutputStream fout;

    try{

        fout = new FileOutputStream (fileName,true);

        int maxLength =
            maxVariableLength(delayDataNumber,
                               countIDs);

        new PrintStream(fout).print("");
        System.out.print(
            "\ndata message prefix |");
        new PrintStream(fout).print(
            "data message prefix |");

        for(int a=0; a<countIDs; a++){

            int space =
                numberSpacesCHAR(delayDataID,
                                   a);

            for(int b=0; b<space; b++){

```

```

        System.out.print(" ");
        new PrintStream(fout).print(
            " ");
    }

    System.out.print(delayDataID[a]);
    new PrintStream(fout).print(
        delayDataID[a]);

    for(int c=0;
        c<(maxLength-space-
            delayDataID[a].length()); c++){
        System.out.print(" ");
        new PrintStream(fout).print(" ");
    }

    System.out.print("|");
    new PrintStream(fout).print("|");
}

System.out.println("  avg");
new PrintStream(fout).println("  avg");

    System.out.print("-----");
    new PrintStream(fout).print(
        "-----");

    for(int d=0; d<countIDs; d++){

        int space = numberSpacesCHAR(delayDataID,
            d);

        System.out.print("+");
        new PrintStream(fout).print("+");

        for(int e=0; e<(maxLength); e++){
            System.out.print("-");
            new PrintStream(fout).print("-");
        }

    }

System.out.println("+-----");
new PrintStream(fout).println("+-----");

String id = null;

//for(int f=0; f<6; f++){
    for(int f=0; f<4; f++){

        if(f==0){
            id="data message size  |";
        }
    }

```

```

        if(f==1){
            id="data msg size (F)  |";
        }
    if(f==2){
        id="data msg size (D)  |";
    }
    if(f==3){
        id="  RTT value: "+valueOfRTT;
        if(valueOfRTT <999){
            id=id+" ";
        }
        if(valueOfRTT <99){
            id=id+" ";
        }
        if(valueOfRTT <9){
            id=id+" ";
        }
        id=id+"  |";
    }
    if(f==4){
        id="    T value: "+valueOfT;
        if(valueOfT <999){
            id=id+" ";
        }
        if(valueOfT <99){
            id=id+" ";
        }
        if(valueOfT <9){
            id=id+" ";
        }
        id=id+"  |";
    }
    if(f==5){
        id="  delivery time  |";
    }

    System.out.print(id);
    new PrintStream(fout).print(id);

    double sumOfRTT = 0;

    for(int g=0; g<countIDs; g++){

        int digitSize =
            numberSpacesINT(delayDataNumber,
                            f,
                            g);

        int leadingSpaces =
            calculateLeadingSpaces(
                            maxLength,
                            digitSize);

        int trailingSpaces =
            calculateTrailingSpaces(

```

```

                                                                maxLength,
                                                                digitSize,
                                                                leadingSpaces);

    for (int h = 0; h<leadingSpaces;
        h++){

        System.out.print(" ");
        new PrintStream(fout).print(" ");
    }

    if (f == 3){
        sumOfRTT =
            sumOfRTT +
            delayDataNumber[f][g];
    }

    System.out.print(
        delayDataNumber[f][g]);
    new PrintStream(fout).print(
        delayDataNumber[f][g]);

    for (int i = 0;
        i<trailingSpaces; i++){

        System.out.print(" ");
        new PrintStream(fout).print(" ");
    }

    System.out.print("|");
    new PrintStream(fout).print("|");

    if (f ==3 && (g+1) == countIDs){
        double avg = sumOfRTT/countIDs;
        avg = avg*1000;
        avg = Math.round(avg);
        avg = avg/1000;
        System.out.print(" "+avg);
        new PrintStream(fout).print(" "+
            avg);

        //write to data file
        writeToExportDataFile(
            testCountDisplay,
            scheduleAlgorithm,
            numberDynamicChannels,
            dataFileExport,
            fileName,
            fileOrder,
            avg);

        sumOfRTT = 0;
    }

```

```

        }

    }

    System.out.println();
    new PrintStream(fout).println();
}

System.out.print("-----");
new PrintStream(fout).print(
    "-----");

    for(int j=0; j<(countIDs); j++){

        int space =
            numberSpacesCHAR(delayDataID,
                j);

        //System.out.print("+");
        //new PrintStream(fout).print("+");
        System.out.print("-");
        new PrintStream(fout).print("-");

        for(int k=0; k<(maxLength); k++){
            System.out.print("-");
            new PrintStream(fout).print("-");
        }

    }

    //System.out.println("|");
    //new PrintStream(fout).println("|");
    System.out.println("-----");
    new PrintStream(fout).println("-----");

// Close the output stream
fout.close();
}

//catches any error conditions
catch (IOException e){
    System.err.println ("unable to write to file");
    System.exit(-1);
} //end catch

} //end displayDelayMatrix

//-----
/**
 * The purpose of this method is to find the maximum
 * length of the variables store in the array for proper
 * display on the screen.
 *

```

```

* @param delayDataNumber
* @param countIDs
*
* @return maxLength
*/

private int maxVariableLength(int [][] delayDataNumber,
                               int countIDs){

    int maxLength = 1;

    for (int a = 0; a<countIDs; a++){

        for (int b = 0; b<3; b++){

            if(delayDataNumber[b][a]>10){
                maxLength=2;

                if(delayDataNumber[b][a]>100){
                    maxLength=3;

                    if(delayDataNumber[b][a]>1000){
                        maxLength=4;

                        if(delayDataNumber[b][a]
                            >10000){
                            maxLength=5;
                        }
                    }
                }
            }
        }
    }

    maxLength = maxLength+3;

    return maxLength;

} //end maxVariableLength

//-----
/**
 * The purpose of this method is to determine the
 * number of spaces for proper display on the screen.
 *
 * @param delayDataID
 * @param countIDs
 *
 * @return space
 */

private int numberSpacesCHAR(String [] delayDataID,
                               int countIDs){

```

```

    int space = 0;

    if(delayDataID[countIDs].length()==5){
        space = 0;
    }

    if(delayDataID[countIDs].length()==4){
        space = 1;
    }

    if(delayDataID[countIDs].length()==3){
        space = 1;
    }

    if(delayDataID[countIDs].length()==2){
        space = 2;
    }

    if(delayDataID[countIDs].length()==1){
        space = 3;
    }

    return space;
} //end numberSpacesCHAR

//-----
/**
 * The purpose of this method is to determine the
 * number of spaces for proper display on the screen.
 *
 * @param delayDataNumber
 * @param a
 * @param b
 *
 * @return space
 */

private int numberSpacesINT(int [][] delayDataNumber,
                            int a,
                            int b){

    int space = 0;

    if(delayDataNumber[a][b]>=0){
        space = 1;
    }

    if(delayDataNumber[a][b]>9){
        space = 2;
    }
}

```

```

        if(delayDataNumber[a][b]>99){
            space = 3;
        }

        if(delayDataNumber[a][b]>999){
            space = 4;
        }

        if(delayDataNumber[a][b]>9999){
            space = 5;
        }

        if(delayDataNumber[a][b]>99999){
            space = 6;
        }

        return space;
} //end numberSpacesINT

//-----
/**
 * The purpose of this method is to calculate the
 * number of leading spaces to align the variable in
 * the cell.
 *
 * @param maxLength
 * @param digitSize
 *
 * @return leadingSpaces
 */
private int calculateLeadingSpaces(int maxLength,
                                  int digitSize){

    int leadingSpaces = 0;
    int addTo         = 0;

    leadingSpaces = maxLength - digitSize;

    if(leadingSpaces == 3){
        leadingSpaces = 3;
    }
    if(leadingSpaces == 4){
        leadingSpaces = 3;
    }
    if(leadingSpaces == 5){
        leadingSpaces = 2;
    }

    return leadingSpaces;
} //end calculateLeadingSpaces

```

```

//-----
/**
 * The purpose of this method is to calculate the
 * number of trailing spaces to align the variable in
 * the cell.
 *
 * @param maxLength
 * @param digitSize
 * @param leadingSpaces
 *
 * @return trailingSpaces
 */

private int calculateTrailingSpaces(int maxLength,
                                   int digitSize,
                                   int
                                   leadingSpaces) {

    int trailingSpaces = 0;
    int addTo          = 0;

    trailingSpaces = maxLength - digitSize -
                      leadingSpaces;

    return trailingSpaces;
}

//-----
/**
 * The purpose of this method is to store the data
 * identifiers (prefixes) in the array
 *
 * @param delayDataID
 * @param dataPrefix
 * @param countIDs
 */

private void storeDataIds(String []delayDataID,
                          String dataPrefix,
                          int countIDs){

    String constructPrefix = "";
    int index=0;

    for(int a = 0; a<dataPrefix.length(); a++){
        char tempChar  = dataPrefix.charAt(a);

        if(tempChar == '[' ){

            for(int b=a+1;
                b<dataPrefix.length(); b++){
                char tempChar2 =
                    dataPrefix.charAt(b);

```



```

//figure out prefix
int numb = 0;

while(
    tempDataPiece.charAt (numb)
        !='0' &&
    tempDataPiece.charAt (numb)
        !='1' &&
    tempDataPiece.charAt (numb)
        !='2' &&
    tempDataPiece.charAt (numb)
        !='3' &&
    tempDataPiece.charAt (numb)
        !='4' &&
    tempDataPiece.charAt (numb)
        !='5' &&
    tempDataPiece.charAt (numb)
        !='6' &&
    tempDataPiece.charAt (numb)
        !='7' &&
    tempDataPiece.charAt (numb)
        !='8' &&
    tempDataPiece.charAt (numb)
        !='9') {

    tempID =
        tempID +"" +
        tempDataPiece.charAt (numb);

    numb = numb +1;
}

if (tempID.equals (currentDataId)) {
    countDataBlockSize =
        countDataBlockSize+1;

    if (b<FIXCHANNELS) {
        countDataBlockSizeFix =
            countDataBlockSizeFix +1;
    }
    else {
        countDataBlockSizeDyn =
            countDataBlockSizeDyn +1;
    }

    tempID="";
}

else{
    tempID="";
}
}
}
}

```

```

        delayDataNumber [0][a] = countDataBlockSize;
        delayDataNumber [1][a] = countDataBlockSizeFix;
        delayDataNumber [2][a] = countDataBlockSizeDyn;

        countDataBlockSize      = 0;
        countDataBlockSizeFix    = 0;
        countDataBlockSizeDyn    = 0;

        tempID="";
    }

} //end countStoreDataBlockSize

//-----
/**
 * The purpose of this method is to count the time
 * periods taken for the data block for each data
 * identifiers (prefixes) and store the value in the array
 *
 * @param ChannelTimePeriodMatrix
 * @param delayDataID
 * @param delayDataNumber
 * @param countIDs
 * @param valueOfRTT
 */
private void countStoreRTT(
        String[][] ChannelTimePeriodMatrix,
        String[] delayDataID,
        int [][] delayDataNumber,
        int countIDs,
        int valueOfRTT){

    String currentDataId      = null;
    String tempID              = "";
    int countTimePeriod       = 0;
    boolean foundInTimePeriod = false;

    for(int a=0; a<countIDs; a++){
        currentDataId = delayDataID[a];

        for(int b=0; b<TIMEPERIODS; b++){

            for(int c=0; c<CHANNELS; c++){

                if(ChannelTimePeriodMatrix[c]
                    [b]!=null){
                    String tempDataPiece =
                        ChannelTimePeriodMatrix[c][b];

                    ////figure out prefix
                    int numb =0;

```

```

while(
    tempDataPiece.charAt (numb)
        !='0' &&
    tempDataPiece.charAt (numb)
        !='1' &&
    tempDataPiece.charAt (numb)
        !='2' &&
    tempDataPiece.charAt (numb)
        !='3' &&
    tempDataPiece.charAt (numb)
        !='4' &&
    tempDataPiece.charAt (numb)
        !='5' &&
    tempDataPiece.charAt (numb)
        !='6' &&
    tempDataPiece.charAt (numb)
        !='7' &&
    tempDataPiece.charAt (numb)
        !='8' &&
    tempDataPiece.charAt (numb)
        !='9') {

    tempID =
        tempID +"" +
        tempDataPiece.charAt (numb);

    numb = numb +1;
}

if(tempID.equals (currentDataId)) {
    foundInTimePeriod = true;
    tempID="";
}

else{
    tempID="";}
}

if(foundInTimePeriod){
    countTimePeriod =
        countTimePeriod+1;
}

foundInTimePeriod = false;
}

tempID="";
delayDataNumber [3][a] =
    countTimePeriod*valueOfRTT;
countTimePeriod = 0;
}

} //end countStoreRTT

```

```

//-----
/**
 * The purpose of this method is to calculate and
 * store T
 *
 * @param delayDataNumber
 * @param dataPrefix
 * @param ratioT
 * @param valueOfT
 */

private void calculateStoreT(
        int [][] delayDataNumber,
        int countIDs,
        int ratioT,
        int valueOfT){

    int blockSizeFix = 0;
    int blockSizeDyn = 0;
    int result      = 0;

    for(int a=0; a<countIDs; a++){

        blockSizeFix = delayDataNumber [1][a];
        blockSizeDyn = delayDataNumber [2][a];

        result =
            ((1)*(valueOfT)*(blockSizeFix)*(ratioT))+
            ((ratioT)*(valueOfT)*(blockSizeDyn));
        delayDataNumber [4][a] = result;

        result = 0;

    }

}

}

//-----
/**
 * The purpose of this method is to calculate the
 * delivery time change and display the result
 *
 * @param delayDataID
 * @param delayDataNumber
 * @param originalTransferTime
 * @param countIDs
 * @param desiredDisplayBreaks
 * @param cellSize
 * @param fileName
 * @param graphName
 */

```

```

private void calculateDisplayThroughputChange(
    String [] delayDataID,
    int [][] delayDataNumber,
    int [] originalTransferTime,
    int countIDs,
    double desiredDisplayBreaks,
    int cellSize,
    String fileName,
    String graphName){

double originalDeliveryTime = 0;
double dynamicDeliveryTime = 0;
double result = 0;
int adjust = 0;

FileOutputStream fout;

try{

    fout =
        new FileOutputStream (fileName,true);

    int maxLength =
        maxVariableLength(delayDataNumber,
                           countIDs);

System.out.print(
    "delivery change |");
new PrintStream(fout).print(
    "delivery change |");

for(int a=0; a<countIDs; a++){
originalDeliveryTime =
    originalTransferTime[a];
dynamicDeliveryTime =
    delayDataNumber [5][a];

if (originalDeliveryTime ==
    dynamicDeliveryTime){
    result = 0;
}
else {
    result = (dynamicDeliveryTime/
        originalDeliveryTime);
    result = (result*100);
    result = (100-result);
    result = Math.round(result);
}

if (result<-9){

```

```

        adjust = maxLength-4;
    }

    if (result>-10){
        adjust = maxLength-2;
    }

    if (result>9){
        adjust = maxLength-3;
    }

    for(int b=0; b<(adjust); b++){
        System.out.print(" ");
        new PrintStream(fout).print(" ");
    }

    System.out.print((int)result+"%|");
    new PrintStream(fout).print(
        (int)result+"%|");
}

for (int c=0; c<2; c++){
    System.out.println(" ");
    new PrintStream(fout).println(" ");
}

result = calculateOverallDeliveryChange(
    delayDataNumber,
    originalTransferTime,
    countIDs);

System.out.println(
    "OVERALL delivery change "+
    "is "+result+"%");
new PrintStream(fout).println(
    "OVERALL delivery change "+
    "is "+result+"%");

//output result to a file for plotting
//values on a graph
//valueForGraph(graphName,
//    result);

for (int d=0; d<2; d++){
    System.out.println(" ");
    new PrintStream(fout).println(" ");
}

for (int e=0;
    e<((cellSize*desiredDisplayBreaks)+40);

```

```

        e++){
            System.out.print("*");
            new PrintStream(fout).print("*");
        }

        for (int f=0; f<2; f++){
            System.out.println(" ");
            new PrintStream(fout).println(" ");
        }

        // Close the output stream
        fout.close();

    }

    //catches any error conditions
    catch (IOException e){
        System.err.println (
            "unable to write to file");
        System.exit(-1);
    }//end catch

} //end calculateDisplayThroughputChange

//-----
/**
 * The purpose of this method is to calculate the
 * delivery time change
 *
 * @param delayDataNumber
 * @param originalTransferTime
 * @param countIDs
 *
 * @return result
 */

private double calculateOverallDeliveryChange(
    int [][] delayDataNumber,
    int [] originalTransferTime,
    int countIDs){

    double originalDeliveryTime = 0;
    double dynamicDeliveryTime = 0;
    double result = 0;

    for(int a=0; a<countIDs; a++){
        originalDeliveryTime =
            originalDeliveryTime +
            originalTransferTime[a];

        dynamicDeliveryTime =

```

```

        dynamicDeliveryTime +
        delayDataNumber [5][a];
    }

    result      = (dynamicDeliveryTime/
        originalDeliveryTime);
    result      = (result*100);
    result      = (100-result);
    result      = Math.round(result);

    return result;
}

} //end calculateOverallDeliveryChange

//-----
/**
 * The purpose of this method is to calculate the
 * delivery time
 * (RTT+T)
 *
 * @param delayDataNumber
 * @param dataPrefix
 * @param originalTransferTime
 * @param fileOrder
 */
private void calculateDeliveryTime(
    int [][] delayDataNumber,
    int countIDs,
    int [] originalTransferTime,
    String fileOrder){

    int totalRTT = 0;
    int totalT   = 0;
    int result   = 0;

    for(int a=0; a<countIDs; a++){
        totalRTT = delayDataNumber [3][a];
        totalT   = delayDataNumber [4][a];

        result   = totalRTT + totalT;
        delayDataNumber [5][a] = result;

        if(fileOrder == "BEFORE"){

            originalTransferTime [a] = result;
        }

        result   = 0;
        totalRTT = 0;
        totalT   = 0;
    }
}

```

```

} //end calculateDeliveryTime

//-----
/**
 * The purpose of this method is to export data to a
 * file which will be used in a Microsoft Excel
 * spreadsheet. The file has commas to delimitate
 * between data
 *
 * @param testCountDisplay
 * @param scheduleAlgorithm
 * @param numberDynamicChannels
 * @param exportFileName
 * @param fileName
 * @param fileOrder
 * @param avg
 */

private void writeToExportDataFile(
    int testCountDisplay,
    String scheduleAlgorithm,
    int numberDynamicChannels,
    String exportFileName,
    String fileName,
    String fileOrder,
    double avg) {

    FileOutputStream fout;

    try{

        fout =
            new FileOutputStream (exportFileName,true);

        if(testCountDisplay == 1 &&
            fileOrder == "BEFORE"){

            new PrintStream(fout).println(
                "test#,scheduleAlgorithm,"+
                "fixchannels,dynamic channels,"+
                "avg. time periods");

        }

        new PrintStream(fout).print(
            testCountDisplay+",");

        if (numberDynamicChannels == 0){

            new PrintStream(fout).print("none,");

        }

        else {

```

```

        new PrintStream(fout).print(
            scheduleAlgorithm+",");
    }

    new PrintStream(fout).print(
        FIXCHANNELS+",");

    new PrintStream(fout).print(
        numberDynamicChannels+",");

    new PrintStream(fout).println(avg);

    // Close the output stream
    fout.close();
}

//catches any error conditions
catch (IOException e){
    System.err.println (
        "unable to write to file");
    System.exit(-1);
} //end catch

} //end writeToExportDataFile
//-----
/**
 * The purpose of this method is to create (append) a
 * file for writing the results to. The results will
 * be used for plotting a graph.
 *
 * @param graphName
 * @param result
 */
private void valueForGraph(String graphName,
                            double result){

    FileOutputStream fout;

    try{

        fout = new FileOutputStream (graphName,true);

        new PrintStream(fout).println(result);

    }

    //catches any error conditions
    catch (IOException e){
        System.err.println (
            "unable to write to file");
        System.exit(-1);
    } //end catch

```

```
}//valueForGraph
```

```
//-----
```

```
}//end class
```

D. PROGRAM – JAVA CLASS: FIRST COME FIRST SERVE

```
/**
```

```
* Filename: FirstComeFirstServe.java  
* Revision: 6 September 2003  
* Date: 18 April 2003  
* Author: Andy Kaminsky  
* Thesis: Channel Allocation  
* Compiler: Java2 SDK 1.4  
*/
```

```
/**
```

```
* The purpose of this class is to allocate future  
* data from a fixed channel to a dynamic channel.  
* The protocol is based on a first come first  
* serve. For example, the first data block  
* will receive all the dynamic channels available.  
*  
* @author: Andy Kaminsky  
*/
```

```
/**
```

```
* Assumptions:  
* (1) There is at least one fixed channel  
* (2) All channels are the same capacity  
* (3) Fixed channels are before the Dynamic  
* channels  
* (4) There is a continuous data block, therefore  
* once a null time period is detected it is  
* assumed that data block is done and no  
* further checking is necessary  
*/
```

```
import java.util.*;
```

```
public class FirstComeFirstServe  
    extends ChannelAllocation {  
  
    public FirstComeFirstServe ( ) { }
```

```
//-----
```

```
/**
```

```

* The purpose of this method is to run through the
* First Come First Serve scheduling algorithm.
*
* @param ChannelTimePeriodMatrix
* @param assignedChannel
*/

protected void firstComeFirstServe(
    String [][] ChannelTimePeriodMatrix,
    String [][] assignedChannel ){

    //count number of dynamic channels
    int numberDynamicChannels =
        countDynamicChannel (assignedChannel);

    //begin at first time period
    //1st FOR LOOP
    for(int currentTimePeriod=0;
        currentTimePeriod < TIMEPERIODS;
        currentTimePeriod++){

        boolean currentChannelFixed = false;
        boolean currentChannelUsed = false;
        String dataID = null;
        int countFutureData = 0;
        int countDown = 0;

        //begin at first channel
        //2nd FOR LOOP
        for(int currentChannel=0;
            currentChannel<CHANNELS;
            currentChannel++){

            //check to see if the channel is
            //fixed AND is used
            if(checkCurrentChannelFixed(
                assignedChannel,
                currentChannel,
                currentChannelFixed) == true
                &&
                checkCurrentChannelUsed (
                    ChannelTimePeriodMatrix,
                    currentChannel,
                    currentTimePeriod,
                    currentChannelUsed) == true){

                //determine the unique data
                //identifier for this data block
                dataID =
                    dataID(ChannelTimePeriodMatrix,
                        currentChannel,
                        currentTimePeriod);

                //count how many time periods of

```

```

//this data block can be changed
countFutureData =
    countFutureData(
        ChannelTimePeriodMatrix,
        currentChannel,
        currentTimePeriod,
        dataID);

//create a temp array and place
//working data block into
String [][] WorkDataBlockMatrix;
WorkDataBlockMatrix =
    new String [3][countFutureData];

//place data block id
workDataMatrix(
    ChannelTimePeriodMatrix,
    assignedChannel,
    WorkDataBlockMatrix,
    currentChannel,
    currentTimePeriod,
    countFutureData);

//find empty dynamic channels in
//time period+1
int emptyDynamicChannel = 0;
int tempTimePeriod = currentTimePeriod;

//find an empty dynamic channel in
//the current time period
for(countDown = countFutureData;
    countDown>0;
    countDown--){

    emptyDynamicChannel =
        findSameSizeEmptyDynamicChannel(
            assignedChannel,
            ChannelTimePeriodMatrix,
            currentChannel,
            tempTimePeriod);

    if(emptyDynamicChannel != 0){
        //assumption for now is data
        //size matches channel
        //capacity
        moveSameSizeEmptyDynamicChannel(
            ChannelTimePeriodMatrix,
            assignedChannel,
            WorkDataBlockMatrix,
            currentChannel,
            tempTimePeriod,
            countFutureData,
            emptyDynamicChannel);
    }
}

```

```

        }

    } // end for

    emptyDynamicChannel = 0;

    //any "NO" in work data matrix needs
    //to be moved back into original array
    anyNO(ChannelTimePeriodMatrix,
        WorkDataBlockMatrix,
        currentChannel,
        tempTimePeriod,
        countFutureData);

    }// end if

} //end 2nd FOR LOOP

} //end 1st FOR LOOP

} // end firstComeFirstServe

//-----
/**
 * The purpose of this method is to count the number
 * of dynamic channels in the assignedChannel matrix
 *
 * @param assignedChannel
 *
 * @return count
 */

private int countDynamicChannel (
    String [][] assignedChannel){

    int count = 0;

    for(int a = 0; a<CHANNELS; a++){
        if(assignedChannel[a][0] == "D"){
            count = count + 1;
        }
    }

    return count;
} //end countDynamicChannel

//-----
/**
 * The purpose of this method is to copy the data
 * block from the ChannelTimePeriodMatrix to a temporary
 * WorkDataMatrix
 *
 * @param ChannelTimePeriodMatrix

```



```

*
* @param ChannelTimePeriodMatrix
* @param WorkDataBlockMatrix
* @param currentChannel
* @param currentTimePeriod
* @param tempNumber
*/

private void moveWorkBackToFixedChannel(
    String [][] ChannelTimePeriodMatrix,
    String [][] WorkDataBlockMatrix,
    int currentChannel,
    int tempTimePeriod,
    int tempNumber){

    ChannelTimePeriodMatrix[currentChannel]
        [tempTimePeriod] =
    WorkDataBlockMatrix[0][tempNumber];

    WorkDataBlockMatrix[2][tempNumber]="YES";

} // end moveWorkBackToFixedChannel

//-----
/**
 * The purpose of this method is to display the
 * contents of the temporary WorkDataMatrix. This
 * helps in troubleshooting
 *
 * @param WorkDataBlockMatrix
 * @param countFutureData
 */

private void displayWorkDataBlock(
    String [][] WorkDataBlockMatrix,
    int countFutureData){

    for(int a = 0; a<countFutureData; a++){

        for(int b = 0; b<3; b++){

            System.out.print(
                WorkDataBlockMatrix[b][a]+" ");

        }

        System.out.println();

    }

} //end displayWorkDataBlock

//-----
/**
 * The purpose of this method is to copy the remaining
 * data blocks from the temporary WorkDataMatrix back

```

```

* to the ChannelTimePeriodMatrix when there are no
* available dynamic channels
*
* @param ChannelTimePeriodMatrix
* @param currentChannel
* @param workTimePeriod
*/

private void moveRemainingDataBlock(
    String [][] ChannelTimePeriodMatrix,
    int currentChannel,
    int workTimePeriod){

    ChannelTimePeriodMatrix[currentChannel]
        [workTimePeriod-1] =
    ChannelTimePeriodMatrix[currentChannel]
        [workTimePeriod];

} //end moveRemainingDataBlock

//-----
/**
* The purpose of this method is to ensure consecutive
* data elements are in correct order over the time
* periods.
*
* @param ChannelTimePeriodMatrix
* @param assignedChannel
* @param currentChannel
* @param currentTimePeriod
*/

private void specialSwapDataCase(
    String [][] ChannelTimePeriodMatrix,
    String [][] assignedChannel,
    int currentChannel,
    int currentTimePeriod){

    //[0] for the data element (ex. h3, e0)
    //[1] for the type of channel (dynamic or fixed)
    //[2] for the data element prefix (ex. h, e)
    //[3] for the value of the data element (ex. 3, 0)
    String [][] SpecialCaseMatrix;
    SpecialCaseMatrix = new String [4][CHANNELS];

    //fill [0]
    for(int a=0; a<CHANNELS; a++){

        //fill [0]
        SpecialCaseMatrix[0][a] =
            ChannelTimePeriodMatrix[a]
                [currentTimePeriod];

        //fill [1]

```

```

SpecialCaseMatrix[1][a] =
    assignedChannel[a][0];

//fill [2]
// determine the unique data identifier for
//this data block
if (SpecialCaseMatrix[0][a] != null){

    String tempID = "";
    String tempDataPiece =
        SpecialCaseMatrix[0][a];

    //figure out prefix
    int numb = 0;

    while(tempDataPiece.charAt(numb) != '0' &&
        tempDataPiece.charAt(numb) != '1' &&
        tempDataPiece.charAt(numb) != '2' &&
        tempDataPiece.charAt(numb) != '3' &&
        tempDataPiece.charAt(numb) != '4' &&
        tempDataPiece.charAt(numb) != '5' &&
        tempDataPiece.charAt(numb) != '6' &&
        tempDataPiece.charAt(numb) != '7' &&
        tempDataPiece.charAt(numb) != '8' &&
        tempDataPiece.charAt(numb) != '9'){

        tempID = tempID + "" +
            tempDataPiece.charAt(numb);
        numb = numb + 1;

    } //end while

    SpecialCaseMatrix[2][a] = tempID;
} //end if

//fill[3]
if (SpecialCaseMatrix[0][a] != null){

    String tempID = "";
    String tempDataPiece =
        SpecialCaseMatrix[0][a];

    //figure out number

    for(int b=0; b<tempDataPiece.length();
        b++){

        if(tempDataPiece.charAt(b) == '0' ||
            tempDataPiece.charAt(b) == '1' ||
            tempDataPiece.charAt(b) == '2' ||
            tempDataPiece.charAt(b) == '3' ||
            tempDataPiece.charAt(b) == '4' ||
            tempDataPiece.charAt(b) == '5' ||
            tempDataPiece.charAt(b) == '6' ||
            tempDataPiece.charAt(b) == '7' ||

```

```

        tempDataPiece.charAt(b)=='8' ||
        tempDataPiece.charAt(b)=='9' {

            tempID = tempID +"" +
                tempDataPiece.charAt(b);

        }
    }//end for

    SpecialCaseMatrix[3][a] = tempID;
} //end if
}

for (int c=0; c<CHANNELS; c++){

    if(SpecialCaseMatrix[1][c]=="F"){

        if(SpecialCaseMatrix[2][c]!=null){

            String tempCheckID    =
                SpecialCaseMatrix[2][c];
            String tempCheckValue =
                SpecialCaseMatrix[3][c];

            for(int d=c; d<CHANNELS; d++){

                if(SpecialCaseMatrix[1][d]=="D"){

                    if(tempCheckID.equals
                        (SpecialCaseMatrix[2][d])){

                        String tempCheckValue2 =
                            SpecialCaseMatrix[3][d];

                        int value =
                            Integer.parseInt(
                                tempCheckValue);

                        int value2 =
                            Integer.parseInt(
                                tempCheckValue2);

                        if(value2<value){

                            ChannelTimePeriodMatrix[c]
                                [currentTimePeriod] =
                                SpecialCaseMatrix[0][d];
                            ChannelTimePeriodMatrix[d]
                                [currentTimePeriod] =
                                SpecialCaseMatrix[0][c];

                            d=CHANNELS;
                        }
                    }
                }
            }
        }
    }
}

```

```

        }
    }
}

} //end specialSwapDataCase

//-----
/**
 * The purpose of this method is to copy the data
 * blocks from the temporary WorkDataMatrix back to the
 * ChannelTimePeriodMatrix when there are available
 * dynamic channels of the same size
 *
 * @param ChannelTimePeriodMatrix
 * @param assignedChannel
 * @param WorkDataBlockMatrix
 * @param currentTimePeriod
 * @param countFutureData
 * @param emptyDynamicChannel
 */

private void moveSameSizeEmptyDynamicChannel(
    String [][] ChannelTimePeriodMatrix,
    String [][] assignedChannel,
    String [][] WorkDataBlockMatrix,
    int currentChannel,
    int currentTimePeriod,
    int countFutureData,
    int emptyDynamicChannel){

    for(int a = 0; a<countFutureData; a++){

        if (WorkDataBlockMatrix[2][a] == "NO"
            &&
            ChannelTimePeriodMatrix
            [emptyDynamicChannel]
            [currentTimePeriod+1] == null
            &&
            assignedChannel
            [emptyDynamicChannel][0] == "D"){

            ChannelTimePeriodMatrix
            [emptyDynamicChannel]
            [currentTimePeriod+1] =
            WorkDataBlockMatrix[0][a];

            WorkDataBlockMatrix[2][a] = "YES";
            a = countFutureData;
        }
    }
}

} //end moveSameSizeEmptyDynamicChannel

```

```

//-----
/**
 * The purpose of this method is to check if the
 * current channel is fixed
 *
 * @param assignedChannel
 * @param currentChannel
 * @param currentChannelFixed
 *
 * @return isFixed
 */

private boolean checkCurrentChannelFixed(
    String [][] assignedChannel,
    int currentChannel,
    boolean currentChannelFixed){

    boolean isFixed = false;

    if(assignedChannel[currentChannel][0]=="F"){

        isFixed = true;
    }

    else {

        isFixed=false;
    }

    return isFixed;

}

}

//-----
/**
 * The purpose of this method is to check if the current
 * channel is used
 *
 * @param ChannelTimePeriodMatrix
 * @param currentChannel
 * @param currentTimePeriod
 * @param currentChannelUsed
 *
 * @return isUsed
 */

private boolean checkCurrentChannelUsed(
    String [][] ChannelTimePeriodMatrix,
    int currentChannel,
    int currentTimePeriod,
    boolean currentChannelUsed){

    boolean isUsed = false;

```

```

        if(ChannelTimePeriodMatrix
            [currentChannel]
            [currentTimePeriod]!=null){

            isUsed=true;
        }

        else {

            isUsed=false;
        }

        return isUsed;

} //end checkCurrentChannelUsed

//-----
/**
 * The purpose of this method is to check if the
 * current channel + 2 time periods of the same data
 * block is used. The idea for this is since the current
 * time period is happening at this time it is too late
 * to use the empty channels however any channels in
 * the future time periods may be used (or better
 * termed:  scheduled)
 *
 * @param ChannelTimePeriodMatrix
 * @param currentChannel
 * @param currentTimePeriod
 * @param dataID
 *
 * @return timePeriodCount
 */

private int countFutureData(
    String [][] ChannelTimePeriodMatrix,
    int currentChannel,
    int currentTimePeriod,
    String dataID){

    int timePeriodCount = 0;
    String tempValue     = null;
    char   tempChar;

    for(int a=0;
        (currentTimePeriod+2+a)<TIMEPERIODS; a++){

        // assumption is a continous data block,
        //therefore once a null time period is
        //detected it is assumed that data block
        //is done and no further checking is
        //necessary
        if(ChannelTimePeriodMatrix
            [currentChannel]
            [currentTimePeriod+2+a] != null){

```

```

        //check to see if part of current data
        //block
        tempValue =
            ChannelTimePeriodMatrix
                [currentChannel]
                [currentTimePeriod+2+a];

        tempChar = tempValue.charAt(0);
        tempValue = tempChar + "";

        if (tempValue.charAt(0) ==
            dataID.charAt(0)){

                timePeriodCount++;
            }
        }
    }

    return timePeriodCount;

} //end countFutureData

//-----
/**
 * The purpose of this method is to determine the data
 * message id used in this scheduling algorithm, for
 * example, one data block is z0,z2,z3... and another
 * data message is y1,y2,y3...
 *
 * @param ChannelTimePeriodMatrix
 * @param currentChannel
 * @param currentTimePeriod
 *
 * @return dataID
 */
private String dataID(
    String [][] ChannelTimePeriodMatrix,
    int currentChannel,
    int currentTimePeriod){

    String firstDataBlock = null;
    String dataID = null;
    char firstDataID;

    firstDataBlock = ChannelTimePeriodMatrix
        [currentChannel]
        [currentTimePeriod];

    firstDataID = firstDataBlock.charAt(0);
    dataID = firstDataID + "";

    return dataID;
}

```

```

} //end dataID

//-----
/**
 * The purpose of this method is to find an empty
 * dynamic channel of the same size as data of a time
 * period to be moved
 *
 * @param assignedChannel
 * @param ChannelTimePeriodMatrix
 * @param currentChannel
 * @param currentTimePeriod
 *
 * @return foundMatch
 */

private int findSameSizeEmptyDynamicChannel(
    String [][] assignedChannel,
    String [][] ChannelTimePeriodMatrix,
    int currentChannel,
    int currentTimePeriod){

    int foundMatch = 0;

    for(int a = 0;
        (currentChannel+a)<CHANNELS; a++){

        // if the channel is Dynamic AND same
        //size AND empty
        if((assignedChannel
            [currentChannel+a][0] == "D")
            //    &&
            //    (assignedChannel[currentChannel+a][1] ==
            //    assignedChannel[currentChannel][1])
            &&
            (ChannelTimePeriodMatrix
            [currentChannel+a]
            [currentTimePeriod+1] == null)){

            foundMatch=(a+currentChannel);
            a = CHANNELS+1;
        }
    }

    return foundMatch;
} //end findSameSizeEmptyDynamicChannel

//-----
/**
 * The purpose of this method is to find any "NO" in
 * the array due to not enough empty dynamic channels
 * and place back into the main Channel Time Period
 * matrix. As a result, when future time periods are
 * examined this data block may be reassigned if any

```



```

* data from a fixed channel to a dynamic channel.
* The scheduling algorithm is based on a
* proportional fair distribution from the
* total number of future data message in a given
* time period. For example, a data message of 10
* units and another data message of 5 units will
* get 2/3 of the dynamic channels and 1/3 of the
* dynamic channels, respectively.
*
* @author: Andy Kaminsky
*/

/**
 * Assumptions:
 * (1) There is at least one fixed channel
 * (2) All channels are the same capacity
 * (3) Fixed channels are before the dynamic
 *     channels
 * (4) There is a continuous data block,
 *     therefore once a null time period is
 *     detected it is assumed that data block
 *     is done and no further checking is
 *     necessary
 */

import java.util.*;

public class FairDistribution
    extends ChannelAllocation {

    public FairDistribution ( ) { }

    //-----
    /**
     * The purpose of this method is to run through the
     * Fair Distribution scheduling algorithm.
     *
     * @param ChannelTimePeriodMatrix
     * @param assignedChannel
     */

    protected void fairDistribution(
        String [][] ChannelTimePeriodMatrix,
        String [][] assignedChannel ){

        //find out the number of fixed channels
        int numberOfFixChannel = 0;
        numberOfFixChannel =
            countFixChannel(assignedChannel);

        //find out the number of dynamic channels

```

```

int numberOfDynamicChannel = 0;
numberOfDynamicChannel =
    countDynamicChannel(assignedChannel);

//begin at first time period
//1st FOR LOOP
for(int currentTimePeriod=0;
    currentTimePeriod < TIMEPERIODS;
    currentTimePeriod++){

    boolean currentChannelFixed = false;
    boolean currentChannelUsed = false;
    String dataID = null;
    int countFutureData = 0;
    int countDown = 0;

    //create a temp array for only this
    //time period
    double [][] futureChannelMatrix;
    futureChannelMatrix =
        new double [2][numberOfFixChannel];

    //initialize array
    for(int a = 0; a<numberOfFixChannel; a++){
        for(int b = 0; b<2; b++){
            futureChannelMatrix[b][a] = 0;
        }
    }

    //begin at first channel
    //2nd FOR LOOP
    for(int currentChannel=0;
        currentChannel<CHANNELS;
        currentChannel++){

        //check to see if the channel
        //is fixed AND is used
        if(checkCurrentChannelFixed
            (assignedChannel,
            currentChannel,
            currentChannelFixed) == true
            &&
            checkCurrentChannelUsed
            (ChannelTimePeriodMatrix,
            currentChannel,
            currentTimePeriod,
            currentChannelUsed) == true){

            //determine the unique data
            //identifier for this data block
            dataID =
                dataID(ChannelTimePeriodMatrix,
                    currentChannel,
                    currentTimePeriod);

```

```

        //count how many time periods of
        //this data block can be changed
        countFutureData =
            countFutureData
                (ChannelTimePeriodMatrix,
                 currentChannel,
                 currentTimePeriod,
                 dataID);

        futureChannelMatrix[0]
            [currentChannel] =
                countFutureData;

    }// end if

} //end 2nd FOR LOOP

//calculate total
//(for determining the percentage)
double countAllFoundData =
    countFoundData
        (futureChannelMatrix,
         numberOfFixChannel);

//calculate the portion to allocate
//dynamic channels to
calculateDynamicChannelAllocation
    (futureChannelMatrix,
     countAllFoundData,
     numberOfFixChannel,
     numberOfDynamicChannel);

//move future data messages to current
//time period dynamic channels
moveToDynamic
    (ChannelTimePeriodMatrix,
     assignedChannel,
     futureChannelMatrix,
     numberOfFixChannel,
     currentTimePeriod);

} //end 1st FOR LOOP

} //end fairDistribution

//-----
/**
 * The purpose of this method is to count the number
 * of fixed channels in the assignedChannel matrix
 *
 * @param assignedChannel
 *
 * @return count
 */

```

```

private int countFixChannel (
    String [][] assignedChannel){

    int count = 0;

    for(int a = 0; a<CHANNELS; a++){
        if(assignedChannel[a][0] == "F"){
            count = count + 1;
        }
    }

    return count;

}

}

//-----
/**
 * The purpose of this method is to count the number
 * of dynamic channels in the assignedChannel matrix
 *
 * @param assignedChannel
 *
 * @return count
 */

private int countDynamicChannel (
    String [][] assignedChannel){

    int count = 0;

    for(int a = 0; a<CHANNELS; a++){
        if(assignedChannel[a][0] == "D"){
            count = count + 1;
        }
    }

    return count;

}

}

//-----
/**
 * The purpose of this method is to count the total
 * number of future data blocks found during this
 * pariticular time period
 *
 * @param futureChannelMatrix
 * @param numberOfFixChannel
 *
 * @return count
 */

private double countFoundData(
    double [][] futureChannelMatrix,
    int numberOfFixChannel){

```

```

        double count = 0;

        for(int a = 0; a<numberOfFixChannel; a++){
            count = count+futureChannelMatrix[0][a];
        }

        return count;
    }//end countFoundData

//-----
/**
 * The purpose of this method is to calculate the
 * number of dynamic channels given to each channel
 * needing future data to send.
 *
 * @param futureChannelMatrix
 * @param countAllFoundData
 * @param numberOfFixChannel
 * @param numberOfDynamicChannel
 */
private void calculateDynamicChannelAllocation(
    double [][] futureChannelMatrix,
    double countAllFoundData,
    int numberOfFixChannel,
    int numberOfDynamicChannel){

    double subtotalAllocated = 0;
    int totalChannels = numberOfFixChannel +
        numberOfDynamicChannel;

    for(int a = 0; a<numberOfFixChannel; a++){

        if(futureChannelMatrix[0][a] == 0){
            futureChannelMatrix[1][a] = 0;
        }

        else {

            double figureOut;

            figureOut = futureChannelMatrix[0][a];
            figureOut =
                Math.round(
                    (figureOut/countAllFoundData)*
                    (totalChannels-numberOfFixChannel));

            futureChannelMatrix[1][a] = figureOut;

            if((figureOut+subtotalAllocated)>
                (totalChannels-numberOfFixChannel)){
                figureOut = figureOut-1;
            }
        }
    }
}

```



```

        double [][] futureChannelMatrix,
        int numberOfFixChannel){

    for(int a = 0; a<numberOfFixChannel; a++){

        if (futureChannelMatrix[0][a]<10){
            System.out.print(
                " "+futureChannelMatrix[0][a]+"| ");
        }

        else {
            System.out.print(
                " "+futureChannelMatrix[0][a]+"|");
        }
    }

    System.out.println();

    for(int b = 0; b<numberOfFixChannel; b++){

        if (futureChannelMatrix[1][b]<10){
            System.out.print(
                " "+futureChannelMatrix[1][b]+"| ");
        }

        else {
            System.out.print(
                " "+futureChannelMatrix[1][b]+"|");
        }
    }

    System.out.println("\n\n");

    return;

} //end displayFutureChannelMatrix

//-----
/**
 * The purpose of this method is to move a data block
 * to a dynamic channel and push the remaining data
 * blocks up one where there is a gap (a queue).
 *
 * @param ChannelTimePeriodMatrix
 * @param assignedChannel
 * @param futureChannelMatrix
 * @param numberOfFixChannel
 * @param currentTimePeriod
 *
 */

private void moveToDynamic(
    String [][] ChannelTimePeriodMatrix,
    String [][] assignedChannel,

```

```

        double [][] futureChannelMatrix,
        int numberOfFixChannel,
        int currentTimePeriod){

for(int currentFixedChannel = 0;
    currentFixedChannel<numberOfFixChannel;
    currentFixedChannel++){

    if (futureChannelMatrix[1]
        [currentFixedChannel]!=0){

        for(int dynamicChannelAllocated=0;
            dynamicChannelAllocated<
                (futureChannelMatrix[1]
                    [currentFixedChannel]);
            dynamicChannelAllocated++){

            double numberToMoveLeft =
                futureChannelMatrix[1]
                    [currentFixedChannel];

            double numberFixedDataLeft =
                futureChannelMatrix[0]
                    [currentFixedChannel];

            //find an empty dynamic channel
            int dynamicChannelLocation =
                findDynamicChannel(
                    ChannelTimePeriodMatrix,
                    assignedChannel,
                    currentTimePeriod);

            //move data element from future
            //fixed to dynamic channel
            moveFixedtoDynamic(
                ChannelTimePeriodMatrix,
                dynamicChannelLocation,
                currentTimePeriod,
                currentFixedChannel);

            //move all remaining data elements
            //of fixed channel up by 1
            moveRemainingFixedUp(
                ChannelTimePeriodMatrix,
                futureChannelMatrix,
                currentTimePeriod,
                currentFixedChannel,
                numberFixedDataLeft);

        }

    }

return;

```

```

} //end moveToDynamic

//-----
/**
 * The purpose of this method is to find an empty
 * dynamic channel in the current time period of the
 * ChannelTimePeriodMatrix
 *
 * @param ChannelTimePeriodMatrix
 * @param assignedChannel
 * @param currentTimePeriod
 *
 */

private int findDynamicChannel(
    String [][] ChannelTimePeriodMatrix,
    String [][] assignedChannel,
    int currentTimePeriod){

    int foundEmptyDynamicChannel = 0;

    for(int a = 0; a<CHANNELS; a++){

        //first find a dynamic channel
        if (assignedChannel[a][0] == "D"){

            //second find an empty dynamic channel
            if(ChannelTimePeriodMatrix
                [a][currentTimePeriod+1] == null){

                //record the match
                foundEmptyDynamicChannel = a;
                a = CHANNELS;
            }
        }
    }

    return foundEmptyDynamicChannel;
} //end findDynamicChannel

//-----
/**
 * The purpose of this method is to move the data
 * element from the fixed channel to the dynamic channel
 *
 * @param ChannelTimePeriodMatrix
 * @param dynamicChannelLocation
 * @param currentTimePeriod
 * @param currentFixedChannel
 *
 */

private void moveFixedtoDynamic(
    String [][] ChannelTimePeriodMatrix,

```

```

        int dynamicChannelLocation,
        int currentTimePeriod,
        int currentFixedChannel){

    ChannelTimePeriodMatrix[dynamicChannelLocation]
        [currentTimePeriod+1]=
        ChannelTimePeriodMatrix[currentFixedChannel]
            [currentTimePeriod+2];

    ChannelTimePeriodMatrix
        [currentFixedChannel]
        [currentTimePeriod+2]=null;

    return;

} //end moveFixedtoDynamic

//-----
/**
 * The purpose of this method is to move the remaining
 * data blocks up by one (to fill in the empty time
 * period left by moving a data element to a dynamic
 * channel)
 *
 * @param ChannelTimePeriodMatrix
 * @param futureChannelMatrix
 * @param currentTimePeriod
 * @param currentFixedChannel
 * @param numberFixedDataLeft
 */
private void moveRemainingFixedUp(
    String [][] ChannelTimePeriodMatrix,
    double [][] futureChannelMatrix,
    int currentTimePeriod,
    int currentFixedChannel,
    double numberFixedDataLeft){

    int lastDataBlock = 0;
    double numberOfDataBlocks =
        futureChannelMatrix[0][currentFixedChannel];

    if(numberFixedDataLeft>1){

        for(int a= 0; a<(numberOfDataBlocks-1);
            a++){

            if(ChannelTimePeriodMatrix
                [currentFixedChannel]
                [currentTimePeriod+a+2] == null){

                ChannelTimePeriodMatrix
                    [currentFixedChannel]
                    [currentTimePeriod+a+2] =

```

```

        ChannelTimePeriodMatrix
            [currentFixedChannel]
            [currentTimePeriod+a+3];

        ChannelTimePeriodMatrix
            [currentFixedChannel]
            [currentTimePeriod+a+3] = null;

        lastDataBlock = a;
    }
}

ChannelTimePeriodMatrix
    [currentFixedChannel]
    [currentTimePeriod+lastDataBlock+3]
    = null;
}

else{

    ChannelTimePeriodMatrix
        [currentFixedChannel]
        [currentTimePeriod+2] = null;
}

return;

} //end moveRemainingFixedUp

//-----
/**
 * The purpose of this method is to check if the
 * current channel is fixed
 *
 * @param assignedChannel
 * @param currentChannel
 * @param currentChannelFixed
 *
 * @return isFixed
 */

private boolean checkCurrentChannelFixed(
    String [][] assignedChannel,
    int currentChannel,
    boolean currentChannelFixed){

    boolean isFixed = false;

    if(assignedChannel[currentChannel][0]=="F"){

        isFixed = true;
    }

    else isFixed=false;
}

```

```

        return isFixed;

} //end checkCurrentChannelFixed

//-----
/**
 * The purpose of this method is to check if the
 * current channel is used
 *
 * @param ChannelTimePeriodMatrix
 * @param currentChannel
 * @param currentTimePeriod
 * @param currentChannelUsed
 *
 * @return isUsed
 */

private boolean checkCurrentChannelUsed(
        String [][] ChannelTimePeriodMatrix,
        int currentChannel,
        int currentTimePeriod,
        boolean currentChannelUsed) {

    boolean isUsed = false;

    if(ChannelTimePeriodMatrix
        [currentChannel]
        [currentTimePeriod]!=null) {

        isUsed=true;
    }

    else isUsed=false;

    return isUsed;

} //end checkCurrentChannelUsed

//-----
/**
 * The purpose of this method is to check if the
 * current channel + 2 time periods of the same data
 * block is used. The idea for this is since the current
 * time period is happening at this time it is too late
 * to use the empty channels, however any channels in
 * the future time periods may be used (or better
 * termed:  scheduled)
 *
 * @param ChannelTimePeriodMatrix
 * @param currentChannel
 * @param currentTimePeriod
 * @param dataID
 *
 * @return timePeriodCount
 */

```



```

private String dataID(
    String [][] ChannelTimePeriodMatrix,
    int currentChannel,
    int currentTimePeriod){

    String firstDataBlock = null;
    String dataID          = null;
    char   firstDataID;

    firstDataBlock =
        ChannelTimePeriodMatrix[currentChannel]
                                [currentTimePeriod];

    firstDataID     = firstDataBlock.charAt(0);
    dataID          = firstDataID + "";

    return dataID;

} //end dataID

//-----

} //end class

```

LIST OF REFERENCES

- [1] Muller, Nathan J. *Desktop Encyclopedia of Telecommunications*. pp. 348. McGraw & Hill, 1998.
- [2] Lundy, G. M. "The Internet." Naval Postgraduate School, Monterey, CA, February 2003.
- [3] Sanders, Ray W. "Bandwidth-On-Demand Layer 1.5 Protocols for Enhanced Broadband Wireless Access System Performance." pp. 1599-1602. *IEEE Communications Magazine*, 2000.
- [4] Jordon, Scott. "Resource Allocation in Wireless Networks". 1996. <www.eng.uci.edu/~sjordan/research/projects/DCA/overview.pdf>, February 2003.
- [5] Welin, Glen. "Internet Connections and Bandwidth Problems". <<http://www.mala.bc.ca/~soules/media112/zine/glen/glen.htm#Bandwidth>>, March 2003.
- [6] Anonymous. "How do you Know Whether Your LAN Needs an Upgrade?". <<http://www.greenet.ge/solutions/upgrade.html>>, March 2003.
- [7] Katz, Randy H. "CS 294-7: Media Access – Aloha and CSMA". <<http://www.sss-mag.com/pdf/1mediaaccess.pdf>>. University of California, Berkeley, CA, 1996.
- [8] Bambos, Nicholas, Chen, Shou, Kim, Jung-Won, Mitra, Debasis. "Noninvasive Channel Probing for Distributed Admission Control and Channel Allocation in Wireless Networks". Stanford University, Stanford, CA, 2002.
- [9] Jordan, Scott and Schwabe, Eric. "Worst-case Performance of Cellular Channel Assignment Policies". pp. 265-275. J.C. Baltzer AG, Science Publishers, 1996.
- [10] Anonymous. "Channel Allocation Problem – Feb. 2002". <www-lce.eng.cam.ac.uk/~shw23/research/Channel%20Allocation>. November 2002.
- [11] Leung, Kin K. and Srivastava, Arty. "Dynamic Allocation of Downlink and Uplink Resource for Broadband Services in Fixed Wireless Networks". Pp. 990-1006. *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 5, May 1999.
- [12] Yardi, Shirang. "Channel Allocation Problem". <www.cs.uga.edu/~rwr/F01_6610/ABSTRACTS/yardi.htm>, December 2002.
- [13] Anonymous. "ICS 161: Design and Analysis of Algorithms – March 1996". <www.ics.uci.edu/~eppstein/161/960312.html>. December 2002.
- [14] Jiang, Hua, Lee, Hee, and Basu, Kalyan. "Self-engineered Adaptive Channel Allocation". pp. 246-250. *IEEE*, March 1999.

- [15] Chen, Jie, Seah, David, and Xu, Wen. "Channel Allocation for Cellular Networks Using Heuristic Methods".
- [16] Ogawa, Takako. "Integrated Service Digital Network", 1999. <<http://xena.fullerton.edu/~ogawat/>>, February 2003.
- [17] Anonymous. "Inverse Multiplexing". <http://whatis.techtarget.com/definition/0,,sid9_gci214323,00.html>, February 2003.
- [18] Wong, Robert Jr., Posey, Melanie, and Becker, Ralph. "An Introduction to ISDN". <<http://america3.pcs.cnu.edu/~dtranngu/paper2.html>>, February 2003.
- [19] Fredette, Paul H. "The Past, Present, and Future of Inverse Multiplexing." pp 42-46. *IEEE Communications Magazine*, April 1994.
- [20] Muller, Nathan J. *Desktop Encyclopedia of Telecommunications*. pp. 241-243. McGraw & Hill, 1998.
- [21] Schrader, Ray. "ISDN". <<http://disc.cba.uh.edu/~rhirsch/spring97/schrad1.htm>>, February 2003.
- [22] Taylor, Steve and Wexler, Joanie. "Link Layer Inverse Multiplexing". 2002. <www.nwfusion.com/newsletters/frame/2002/01674487.html>, February 2003.
- [23] 3Com. "Inverse Multiplexing over ATM (IMA)". 3Com Technical Papers, 1997.
- [24] Sklower, K., Lloyd, B., McGregor, G., Carr, D., Coradetti, T. "RFC 1990, The PPP Multilink Protocol". 1996. <<http://www.armware.dk/RFC/rfc/rfc1990.html>>, March 2003.
- [25] Anonymous. "RFC 1618 – PPP over ISDN". <<http://asg.web.cmu.edu/rfc/rfc1618.html>>, May 2003.
- [26] Anonymous. "All about Narrowband ISDN". <<http://www.harvard.edu/~fine/ISDN/n-isdn.html>>, April 2003.
- [27] Snoeren, Alex C. "Adaptive Inverse Multiplexing for Wide-Area Wireless Networks". Massachusetts Institute of Technology, December 1999.
- [28] Fountanas, Leonidas. "An Assessment of Emerging Wireless Broadband Technologies". Thesis. Naval Postgraduate School, Monterey, CA December 2001.
- [29] Anonymous. "Synchronous Adaptive Infrastructure Network (SAIN) Background". <<http://www.circuitpath.com/technology/background.htm>>, May 2003.

- [30] Ha, Joon-Ho and Pinkston, Timothy. "A Token-based Channel Access Protocol for Wavelength Division Multiplexed Optically Interconnected Multiprocessors." <<http://ipdps.eece.unm.edu/1997/wocs/hapinkst.pdf>>, University of Southern California, March 2003.
- [31] Cu, Gregory and Marcos, Nelson. "CSMA/CD-Based Multiple Network Lines Dynamic Utilization Algorithm". pp. 151-160. Proceedings of the Philippine Computing Science Congress, 2000.
- [32] Hac, Anna and Chen, Zhengping. "Hybrid Channel Allocation in Wireless Networks". pp. 2329-2333. *IEEE Communications Magazine*, 1999.
- [33] Anonymous. Linux Networx. <http://www.linuxnetworx.com/products/dual_eth.php>, March 2003.
- [34] Anonymous. Beowulf Page, University of Southern California. <<http://ilab.usc.edu/beo>>, March 2003.
- [35] Zhao, Baosong and Andersen, Daniel. "Heterogeneous Channel Bonding on a Beowulf Cluster". < <http://www.dvo.ru/bbc/pdpta/vol5/p410.pdf>>, March 2003.
- [36] Qiao, Daji and Shin, Kang G. "Achieving Efficient Channel Utilization and Weighted Fairness for Data Communications in IEEE 802.11 WLAN under the DCF". University of Michigan.
- [37] Tanenbaum, Andrew S. "*Computer Networks*". pp. 214-215, 631-641. Prentice-Hall, Inc., 1998.
- [38] Kurose, James F. and Ross, Keith W. "*Computer Networking*". pp. 18-19. Addison Wesley, 2001.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California
3. Professor John Gibson
Department of Computer Science
Naval Postgraduate School
Monterey, California
4. Professor Geoffrey Xie
Department of Computer Science
Naval Postgraduate School
Monterey, California